

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

"На правах рукопису"
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“ ” _____ 2019р.

Магістерська дисертація

зі спеціальності 121 Інженерія програмного забезпечення
за спеціалізацією Інженерія програмного забезпечення розподілених систем
на тему Веб-додаток інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”

Виконав (-ла): студент (-ка) 6 курсу, групи ТВ-81мп
Амброс Світлана Миколаївна
(прізвище, ім'я, по батькові)

_____ (підпис)

Науковий керівник доц. к.т.н. Ковальчук А.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

Рецензент _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ - 2019

“Київський політехнічний інститут ім. Ігоря Сікорського”

за спеціалізацією - Інженерія програмного забезпечення розподілених систем

«_____» _____ 2019p.

НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

6. Орієнтований перелік ілюстративного матеріалу високорівнева архітектура системи, загальна архітектура веб-додатку, діаграма класів, діаграма збірки проекту з використанням webpack, діаграма послідовностей роботи додатку, ілюстрації роботи додатку

7. Орієнтований перелік публікацій 1. Амброс С.М., Ковальчук А. М. Веб-додаток інтернет сервісу “Відкритий спорт-майданчик з е-сервісами”// Збірник тез VI Всеукраїнської науково-практичної конференції молодих науковців 16 травня 2019 р., м. Київ, Україна. – С. 6.

2. Амброс С.М., Ковальчук А.М. Розробка веб-додатку для відкритого спортивного майданчику з е-сервісами// Матеріали XVII Міжнародної науково-практичної конференції молодих вчених та студентів, 23-26 квітня 2019 року, м. Київ, Україна. – С. 134

8. Дата видачі завдання « ____ » _____ 201 ____ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Строки виконання етапів магістерської дисертації	Примітка
1.	Вивчення та аналіз задачі	02.10.18 – 09.10.18	
2	Розробка архітектури та загальної структури системи	10.10.18 – 24.12.18	
3.	Розробка окремих частин системи	10.02.19 – 23.04.19	
4.	Програмна реалізація системи	24.04.19 – 27.10.19	
5	Проходження переддипломної практики	02.09.19 – 27.10.19	
6.	Оформлення пояснювальної записки	06.09.19 – 15.11.19	
7.	Захист програмного продукту	22.10.19 – 22.10.19	
8.	Передзахист	21.11.19 – 21.11.19	
9.	Захист	16.12.19 – 24.12.19	

Студент

(підпис)

Амброс С.М.

(прізвище та ініціали)

Науковий керівник

(підпис)

Ковальчук А.М.

(прізвище та ініціали)

РЕФЕРАТ

Магістерська дисертація складається зі вступу, п'яти розділів, висновку, переліку посилань з 32 найменування, 4 додатки, і містить 34 рисунки, 22 таблиці. Повний обсяг магістерської дисертації складає 106 сторінок, з яких перелік посилань займає 2 сторінки, додатки — 21 сторінку.

Актуальність теми. Спорт корисний для здоров'я, але часом через навантажений певними проблемами графік, люди не завжди знаходять час зв'язатись з адміністратором або відвідати заклад, щоб записатись на заняття, скасувати заняття чи купувати квитки завчасно. Через це виникають проблеми, коли не бажаючи витратити свій час на велику послідовність дій, відвідувач покидає майданчик, а заклад втрачає клієнта, чи коли через неправильне розподілення людей майданчики або переповнені, або майже пусті. Тому, проблема полягає в розробці інформаційної системи, яка буде спрощувати бізнес процеси спортивного закладу, шляхом надання сервіс-орієнтованих програмних засобів.

Мета та задачі дослідження. Метою даної роботи є створення програмного забезпечення, яке надасть автоматизовані сервіси в сферу контролю та управління доступом спортивних комплексів, що дозволить контролювати навантаження закладу та збереже відвідувачів.

Рішення поставлених завдань та досягнуті результати. Відповідно до вимог та мети роботи було визначено та вирішено основні завдання:

- Визначити архітектуру системи “Відкритого спортивного майданчику”;
- Проаналізувати існуючі системи та рішення;
- Визначити архітектуру та розробити сервіс-орієнтований веб-додаток відкритого спортивного майданчику, який буде швидким, кросплатформенним та надасть актуальну інформацію закладу з можливістю виконувати онлайн основні операції майданчику;
- Розробити концепцію стартап-проекту для можливості його комерціалізації;
- Запропонувати варіанти подальшого покращення чи розширення системи.

Об’єктом дослідження є інформаційні системи забезпечення бізнес процесів.

Предметом дослідження є програмні засоби забезпечення та супроводу сервіс-орієнтованих бізнес процесів спортивних комплексів.

Методи досліджень. Для вирішення завдань було визначено архітектура розроблюваного додатку, а також проаналізовано сучасні підходи до побудови веб-застосунків з використанням мови програмування TypeScript та фреймворку Angular та перевірено їх ефективність на прикладі власного додатку.

Інноваційна новизна роботи полягає у виборі ефективних засобів розробки веб-додатку, що зробить його гнучким для розширення, кросплатформенним та швидким.

Практичне значення одержаних результатів. В ході роботи був розроблений сервіс-орієнтований веб-додаток який надає автоматизовані сервіси в сферу контролю та управління доступом спортивних комплексів.

Апробації результатів дисертації. Результати роботи дисертації було оприлюднено у:

1. VI Всеукраїнської науково-практичної конференції молодих науковців 16 травня 2019 р.
2. XVII Міжнародної науково-практичної конференції молодих вчених та студентів, 23-26 квітня 2019 року

Публікації

Амброс С.М., Ковальчук А. М. Веб-додаток інтернет сервісу “Відкритий спорт-майданчик з е-сервісами”// Збірник тез VI Всеукраїнської науково-практичної конференції молодих науковців 16 травня 2019 р., м. Київ, Україна. – С. 6.

Амброс С.М., Ковальчук А.М. Розробка веб-додатку для відкритого спортивного майданчику з е-сервісами// Матеріали XVII Міжнародної науково-практичної конференції молодих вчених та студентів, 23-26 квітня 2019 року, м. Київ, Україна. – С. 134С.

Ключові слова. Сервіс-орієнтована архітектура, TypeScript, Angular, NPM, Webpack, REST, Клієнт-серверна архітектура, Dependency Injection.

РЕФЕРАТ

Магистерская диссертация состоит из введения, пяти глав, заключения, списка ссылок из 32 наименований, 4 приложения, и содержит 34 рисунка, 22 таблицы. Полный объем магистерской диссертации составляет 106 страниц, из которых перечень ресурсов занимает 2 страницы, приложения - 21 страницы.

Актуальность темы

Спорт полезен для здоровья, но иногда из-за нагруженный определенными проблемами график, люди не всегда находят время связаться с администратором или посетить заведение, чтобы записаться на занятия, сообщить об отмене занятия или покупать билеты в кассах заведения заблаговременно. Из-за этого возникают проблемы, когда не желая тратить свое время на большую последовательность действий, посетитель покидает площадку, а заведение теряет клиента, или когда из-за неправильного распределения людей площадки или переполнены, или почти пустые.

Поэтому, проблема заключается в том, чтобы разработать информационную систему, которая будет упрощать бизнес процессы спортивного заведения, путем предоставления сервис-ориентированных программных средств.

Цель и задачи исследования

Целью данной работы является создание программного обеспечения, которое предоставит автоматизированные сервисы в сферу контроля и управления доступом спортивных комплексов, что позволит контролировать нагрузку заведения и сохранит посетителей.

Решение поставленных задач и достигнутых результатах

В соответствии с требованиями и целями работы были определены и решены основные задачи:

- Определить архитектуру системы “Открытой спортивной площадки”;
- Проанализировать существующие системы и решения;

— Определить архитектуру и разработать сервис-ориентированное веб-приложение открытой спортивной площадки, которое будет быстрым, кроссплатформенным и предоставит актуальную информацию заведения с возможностью выполнять онлайн основные операции площадки;

— Разработать концепцию стартап-проекта для возможности его коммерциализации;

— Предложить варианты дальнейшего улучшения или расширения системы.

Объект исследований

Информационные системы обеспечения бизнес процессов.

Предмет исследований

Программные средства обеспечения и сопровождения сервис-ориентированных бизнес процессов спортивных комплексов.

Методы исследований

Для решения задач были определены архитектура разрабатываемого приложения, а также проанализированы современные подходы к построению веб-приложений с использованием языка программирования TypeScript и фреймворка Angular и проверено их эффективность на примере собственного приложения.

Инновационная новизна

Инновационная новизна работы заключается в выборе эффективных средств разработки веб-приложения, что сделает его гибким для расширения, кроссплатформенным и быстрым.

Практическое значение полученных результатов

В ходе работы был разработан сервис-ориентированное веб-приложение которое предоставляет автоматизированные сервисы в сфере контроля и управления доступом спортивных комплексов.

Апробации результатов диссертации

Результаты работы диссертации были обнародованы в:

1. VI Всеукраинской научно-практической конференции молодых ученых 16

мая 2019;

2. XVIII Международный научно-практической конференции молодых ученых и студентов, 23-26 апреля 2019.

Публикации

Амброс С.М., Ковальчук А. Н. Веб-приложение интернет сервиса “Открытая спорт-площадка с е-сервисами” // Сборник тезисов VI Всеукраинской научно-практической конференции молодых ученых 16 мая 2019, г. Киев, Украина.. - С. 6.

Амброс С.М., Ковальчук А.М. Разработка веб-приложения для открытой спортивной площадки с е-сервисами // Материалы XVIII Международный научно-практической конференции молодых ученых и студентов, 23-26 апреля 2019 года, г. Киев, Украина.. - С. 134с.

Ключевые слова

Сервис-ориентированная архитектура, TypeScript, Angular, NPM, Webpack, REST, клиент-сервер, Dependency Injection.

ABSTRACT

The master's thesis consists of an introduction, five sections, a conclusion, a list of links of 32 titles, 4 annexes, and contains 34 drawings, 22 tables. The full volume of the master's thesis is 106 pages, of which the list of links occupies 2 pages, the annexes - 21 page.

Actuality of theme

Sport is healthy, but sometimes due to busy schedule, people do not always find the time to contact the administrator or visit the playground to register for classes, inform to cancel classes or buy tickets earlier. Therefore, people do not want to spend their time on a big action sequence for making some action with tickets in playground, a visitor leaves the site and ground loses a client, or due to improper distribution platforms ground is overcrowded, or almost empty.

The aims and objectives of the study

The aim of this work is to create software that provides automated services in the access and control field of sports complexes, allowing control load facility and retain visitors.

The solution of tasks and results

According to the requirements was determined and solved the basic problem:

- Define the architecture of “Outdoor playground with e-services” system, analyze existing systems and solutions;
- Define the architecture and develop a service-oriented web application outdoor playground to be fast, cross-platform and provide relevant information institutions with the ability to perform basic operations online site;
- Develop a concept for a startup project for its possible commercialization;

Object of research

Information system of business processes.

Subject of research

Software maintenance and support service-oriented business processes sports complexes.

Methods

For solving problems were identified architecture of developed applications and analyzes current approaches for building Web applications using the programming language and framework TypeScript Angular and tested their efficiency by the example of own application.

Innovative novelty

Innovative novelty is to choose effective ways of web application development, making it flexible for expansion cross-platform and fast.

The practical significance of the results

Was developed service-oriented web application that provides automated services in the access and control field of sports complexes.

Thesis Testing Results

The results of the thesis was published in

1. VI All-Ukrainian scientific conference of young scientists May 16, 2019
2. XVII International scientific conference of young scientists and students, 23-26

April 2019

Publications

Ambros S., Kovalchuk A. Web application Internet service "Outdoor playground with e-services" // Abstracts of the VI All-Ukrainian scientific conference of young scientists May 16, 2019, m. Kyiv, Ukraine. - P. 6.

Ambros S., Kovalchuk A. Development Web Application for outdoor playground with e-services // Proceedings of the XVII International scientific conference of young scientists and students, 23-26 April 2019, m. Kyiv, Ukraine. - S. 134S.

Keywords

Service-oriented architecture, TypeScript, Angular, NPM, Webpack, REST, Client-server architecture, Dependency Injection.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	14
ВСТУП.....	16
1. СИСТЕМИ АВТОМАТИЗАЦІЇ РОБОТИ СПОРТИВНИХ КОМПЛЕКСІВ.....	18
1.1. Існуючі рішення.....	18
1.2. Постановка завдання роботи.....	22
2. ЗАСОБИ ЕФЕКТИВНОЇ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ.....	24
2.1. Система контролю версій Git.....	24
2.2. Менеджер пакетів NPM.....	27
2.3. Мова програмування TypeScript.....	27
2.4. Angular фреймворк.....	28
2.5. Постачальник модулів Webpack.....	29
2.6. Препроцесор стилів Sass.....	32
2.7. Реактивне програмування RxJS.....	34
3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ВІДКРИТОГО СПОРТМАЙДАНЧИКУ.....	37
3.1. Клієнт-серверна архітектура REST.....	37
3.2. Архітектура веб-додатку.....	40
3.3. Модулі веб-додатку.....	43
3.4. Сервіси веб-додатку.....	45
3.5. Компоненти веб-додатку.....	49
3.5.1. Компоненти та їх складові.....	49
3.5.2. Data binding.....	51
4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ.....	54
5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ.....	61
5.1. Опис ідеї стартап-проекту.....	61
5.2. Технологічний аудит ідеї проекту.....	63

5.3. Аналіз ринкових можливостей запуску стартап-проекту	64
5.4. Розробка ринкової стратегії проекту	73
5.5. Розробка маркетингової програми.....	77
5.6. Результати аналізу стартап ідеї	80
ВИСНОВКИ	82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	84
Додаток А	86

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування.

TypeScript (TS) — мова програмування, представлений Microsoft в 2012 році і позиціонується як засіб розробки веб-додатків, що розширює можливості JavaScript.

Фреймворк — інфраструктура програмних рішень, що полегшує розробку складних систем.

Angular — це платформа для створення мобільних та десктопних веб-додатків.

NPM — менеджер пакетів, що входить до складу Node.js.

Webpack — це постачальник модулів JavaScript з відкритим кодом, який він може трансформувати активні компоненти, такі як HTML, CSS та зображення тощо.

GIT — це одна з найвідоміших систем контролю версій з відкритим вихідним кодом, на яку покладаються мільйони проектів по всьому світу.

RxJs — бібліотека для асинхронного програмування.

JetBrains WebStorm — інтегроване середовище розробки на JavaScript, CSS та HTML від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA.

Сервер — у комп'ютерній термінології термін може стосуватися окремого комп'ютера чи програми. Головною ознакою в обох випадках є здатність машини чи програми переважну кількість часу працювати автономно, без втручання людини, реагуючи на зовнішні події відповідно до встановленого програмного забезпечення.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними.

Бібліотека — збірка підпрограм або об'єктів, що використовуються для розробки програмного забезпечення (ПО).

REST (Representational State Transfer – “передача стану”) — архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST є узгоджений набір обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи.

MVC (Model-View-Controller) — шаблон архітектури, який відділяє дані, візуальне відображення та поведінку обробки подій у різні класи, а саме: Модель(Model), Представлення (View) та Контролер (Controller).

Dependency Injection (DI) — дизайн шаблон, що використовується для впровадження IoC. Це дозволяє створювати залежні об’єкти поза класом і надає ці об’єкти класу різними способами. Використовуючи DI, ми переміщуємо створення та зв’язування залежних об’єктів поза класом, який залежить від них.

СКУД – системи контролю та управління доступом.

Сервіс-орієнтована архітектура — проектування та розробки сервісів і засобів їх підключення, кожен з яких являє собою бізнес-функцію, призначену для забезпечення узгодженої роботи додатків.

ВСТУП

Спорт корисний для здоров'я, але часом через навантажений певними проблемами графік, люди не завжди знаходять час зв'язатись з адміністратором або відвідати заклад, щоб записатись на заняття, повідомити заклад про скасування заняття чи купувати квитки в касах закладу завчасно. Через це виникають проблеми, коли не бажаючи витратити свій час на велику послідовність дій, відвідувач покидає майданчик, а заклад втрачає клієнта, чи коли через неправильне розподілення людей майданчики або переповнені, або майже пусті.

Тому, проблема полягає в тому, щоб розробити інформаційну систему, яка буде спрощувати бізнес процеси спортивного закладу, шляхом надання сервіс-орієнтованих програмних засобів.

Отже, актуальним є створення ПЗ, яке надасть автоматизовані сервіси в сферу контролю та управління доступом спортивних комплексів, що дозволить контролювати навантаження закладу та збереже відвідувачів.

Для цього потрібна система відкритого спортивного майданчику, яка має складатись з веб-застосунку, серверу та системи контролю і управління доступом.

Веб-застосунок дозволить додати в систему сервіс-орієнтоване програмне забезпечення, яке надасть користувачеві можливість з будь-якого пристрою маючи Інтернет виконувати онлайн такі операції як реєстрація та авторизація, отримання даних розкладу занять, кількості вільних місць та деталей події, бронювання та перегляд особистих актуальних занять чи скасування їх.

Користувачами системи можуть жителі міста, де встановлена система відкритого спортивного майданчику.

На сьогодні існують схожі рішення, які надають інформацію щодо спортивних установ, наприклад, MyFit, GiftFitness.

Дана система дозволить розподілити та контролювати відвідувачів спортивного закладу за графіком, бронювати чи скасовувати заняття онлайн без

необхідності зв'язування з адміністратором спортивного майданчику, а також зберігатиме особисті дані користувачів, що відповідно скоротить послідовність дій користувачів та дозволить зберегти власний час.

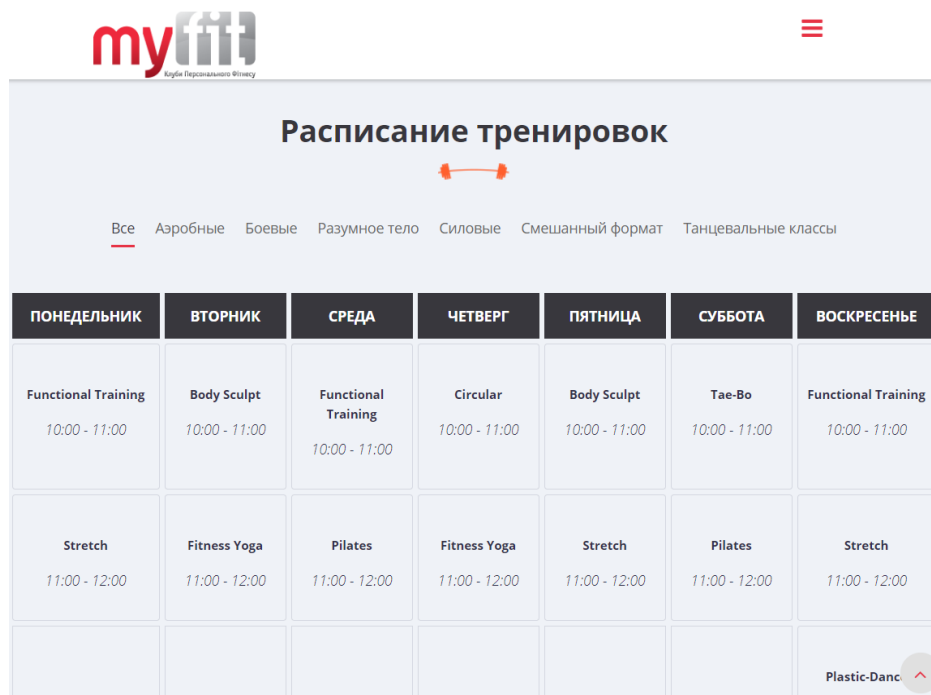
Таким чином розроблювана система допоможе контролювати та керувати потоком відвідувачів майданчику, правильно розподіляти завантаження майданчику, та спрощувати послідовність дій, яку має зробити користувач для того щоб стати відвідувачем певного спортивного закладу.

1. СИСТЕМИ АВТОМАТИЗАЦІЇ РОБОТИ СПОРТИВНИХ КОМПЛЕКСІВ

Через наявність в спортивних закладів проблеми доступу відвідувачів до майданчику, існують інформаційні системи, що надають автоматизацію контролю та управління доступом спортивних комплексів. Було проаналізовано існуючі рішення та визначено вимоги до створюваної системи. Також, було визначено завдання для даної дипломної роботи.

1.1. Існуючі рішення

На даний момент існують системи, що надають інформацію щодо спортивних закладів, наприклад “MyFit” , “Gift Fitness” (рисунок 1.1, 1.2).



ПОНЕДЕЛЬНИК	ВТОРНИК	СРЕДА	ЧЕТВЕРГ	ПЯТНИЦА	СУББОТА	ВОСКРЕСЕНЬЕ
Functional Training 10:00 - 11:00	Body Sculpt 10:00 - 11:00	Functional Training 10:00 - 11:00	Circular 10:00 - 11:00	Body Sculpt 10:00 - 11:00	Tae-Bo 10:00 - 11:00	Functional Training 10:00 - 11:00
Stretch 11:00 - 12:00	Fitness Yoga 11:00 - 12:00	Pilates 11:00 - 12:00	Fitness Yoga 11:00 - 12:00	Stretch 11:00 - 12:00	Pilates 11:00 - 12:00	Stretch 11:00 - 12:00
						Plastic-Danc

Рисунок 1.1 — Система “MyFit”

Вони надають можливість перегляду розкладу, видів спорту, які викладаються, але не надають можливості онлайн записатись чи скасувати заняття.

Зазвичай, для того щоб записатись необхідно телефонувати адміністратору. В додаток до цього, заняття можуть переноситись лише обмежену кількість разів, а випадку не переносу заняття, відвідувач його втрачає, а відповідно і кошти, які було за заняття заплачено також, що може не задовільнити відвідувача та призвести до вибору іншого спортивного закладу. Крім того користувачеві не завжди зручно телефонувати про відміну заняття, через це у спортивного майданчику може бути неактуальна інформація щодо кількості відвідувачів в певний час, а тому заклад може бути або переповнений, або пустий. Тому проаналізувавши дані системи було визначено базові вимоги до створюваної системи.

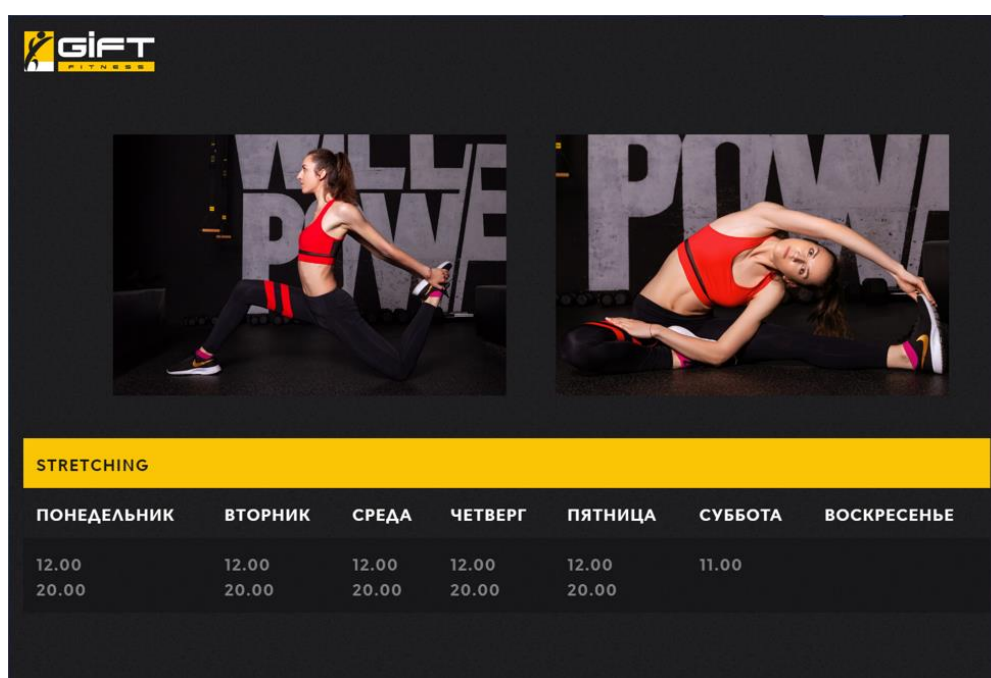


Рисунок 1.2 — Система “ Gift Fitness ”

Оскільки на території міста є багато спортивних майданчиків, які є відкритими та можуть бути оновленими та покращеними, було вирішено створити систему “Відкритого спортивного майданчику”, яка б дозволяла людям відвідувати майданчик у будь-який час, дозволяла переглядати інформацію щодо заходів та наявності місць, дозволяла б записуватись на подію чи скасовувати запис онлайн. На даний момент проект вирішено реалізувати на прикладі спортивного майданчику на території студентського містечка.

Оскільки система повинна бути гнучкою та розширюваною при побудові

архітектури відкритого спортивного майданчику було прийнято рішення зробити її сервіс-орієнтованою.

Сучасні підприємства, заклади і спортивні заклади в тому числі, потребують швидкого реагування на зміну бізнес цілей, гнучку інтеграцію та збереження працездатності закладу з підтримкою бізнесу в реальному часі. Для даних цілей добре підходить сервіс-орієнтована архітектура.

Сервіс-орієнтована архітектура будується за рахунок проектування та розробки сервісів і засобів їх підключення. Сервіс являє собою певну роботу або бізнес-функцію, призначену для забезпечення узгодженої роботи додатків[1].

Поява сервіс-орієнтованої архітектури дозволяє перейти від програмування комплексних інформаційних продуктів до можливості забезпечення інформаційних потреб системи з різнорідних додатків. Так само не маловажним аспектом є висока гнучкість, яка досягається за рахунок можливості швидкого коректування бізнес-логіки — зміна, що вноситься в бізнес-функцію, в результаті торкнеться всі необхідні додатки[2].

Відповідно до сервіс-орієнтованої архітектури, було розроблено високорівнева архітектура системи відкритого спортивного майданчику (рисунк 1.3), яка відображає розбиття загальної системи на окремі сервіси відповідно до їх бізнес функцій, такі як сервіс контролю та управління доступом, платіжна система, система відеоспостереження, система суддівства, система видачі спортивного інвентаря, система енергопостачання та система вуличного освітлення. Це показує, що відповідно до принципів сервіс-орієнтованих систем, сервіси створюються окремо, розширюючи поступово існуючий варіант системи, та при зміні бізнес-вимог зміни не торкнуться всієї системи, а лише певних її сервісів. Також, система може працювати навіть з наявністю базового сервісу контролю та управління доступу, а інші сервіси, які будуть розроблюватись далі, не зупиняють можливість введення в експлуатацію системи зараз, а лише розширять та додадуть нові функції для майданчику згодом, приваблюючи відвідувачів ще більше.

Визначивши проблему існуючих систем та проаналізувавши приклади застосунків(рисунк 1.1, 1.2) було визначено вимоги до клієнтської частини. Такі як:

надання актуальної інформації користувачеві, надання можливості виконувати базові операції майданчику онлайн, кросплатформеність та швидкодія. Відповідно до даних вимог було вирішене створити клієнтську частину як веб-застосунок, що вирішить проблему кросплатформеності та швидкодії, а для можливості виконання операцій онлайн та забезпечення актуальної інформації, веб-застосунок комунікуватиме з сервером, який пов'язаний з системою СКУД.

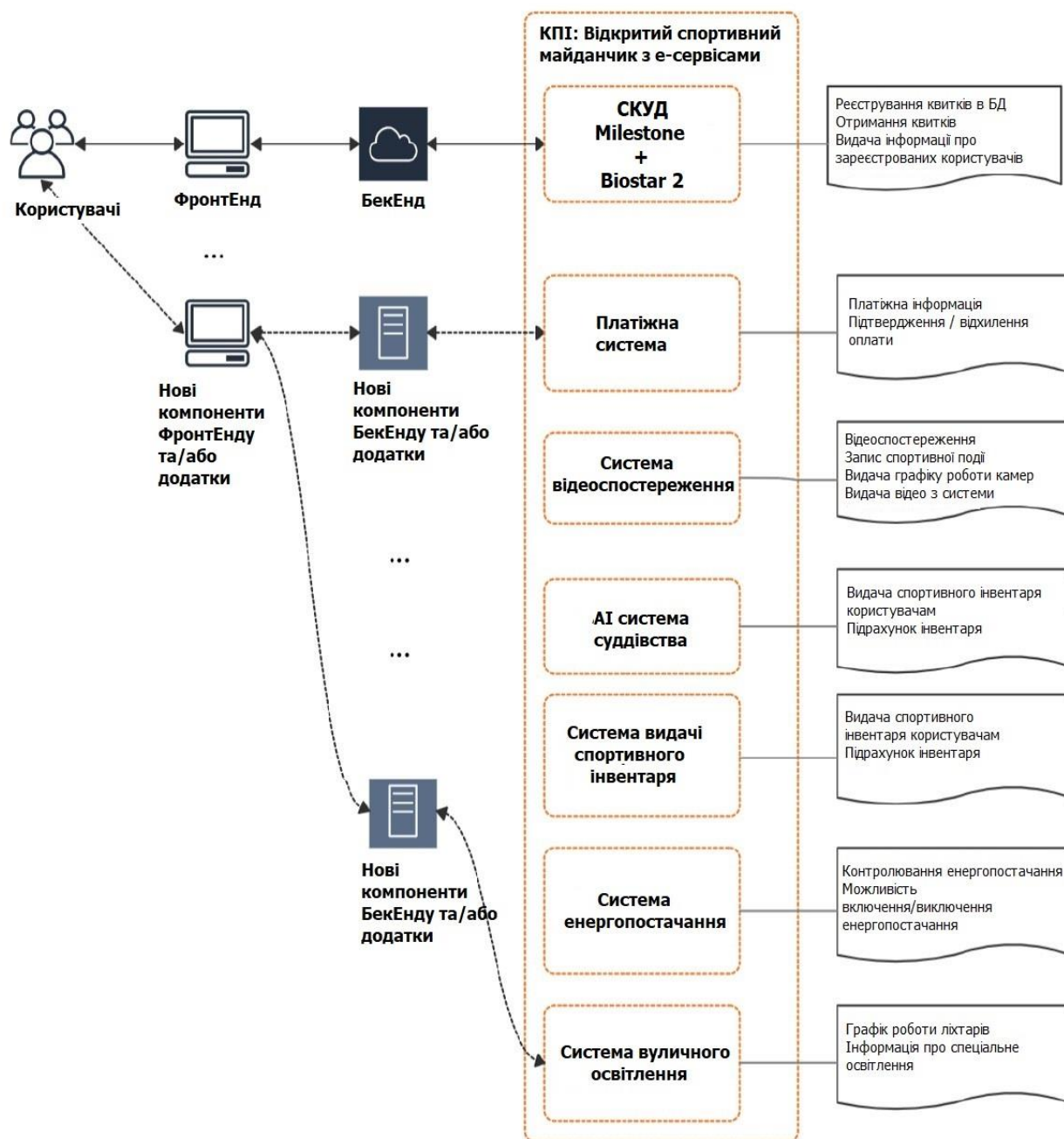


Рисунок 1.3 — Архітектура системи

Отже, інформаційна система контролю і управління доступом складається з 3 основних компонентів(рисунок 1.3): Front-end додаток, розроблений на Angular фреймворці, Back-end додаток розгорнутий в хмарі та система СКУД.

На даний момент розроблювалась базова інформаційна система контролю і управління доступом, яка складаються з Front-end додатку, Back-end додатку та системи СКУД, які виконують такі функції як реєстрування квитків в БД, отримання квитків, видача інформації про зареєстрованих користувачів. Надалі система може розширюватись да доповнюватись сервісами відеонагляду, енергопостачання, системою спортивного інвентаря, платіжної системами та іншими.

1.2. Постановка завдання роботи

Метою даної роботи є створення програмного забезпечення, яке надасть автоматизовані сервіси в сферу контролю та управління доступом спортивних комплексів, що дозволить контролювати навантаження закладу та збереже відвідувачів.

Додаток розроблений як веб-сайт, тому доступний користувачам при наявності мережі Інтернет з десктопу, планшету та мобільного пристрою.

Користувачем системи можуть бути студенти та жителі міста, де знаходиться майданчик.

Вхідними даними є особисті дані користувача, а саме пошта, ім'я, прізвище та студентський квиток, якщо це студент.

Вихідними даними є дані, щодо заброньованих заходів.

В ході виконання роботи було визначено наступні вимоги до системи:

- Надання актуальної інформацію користувачеві;
- Виконання базових операції майданчику онлайн – авторизація, реєстрація, перегляд розкладу, бронювання, скасування бронювання;
- Кросплатформенність та швидкодія.

Відповідно до вимог та мети роботи було визначено основні завдання:

- Визначити архітектуру системи “Відкритого спортивного майданчику”;
- Проаналізувати існуючі системи та рішення;
- Визначити архітектуру веб-додатку та розробити сервіс-орієнтований веб-додаток відкритого спортивного майданчику, який буде швидким, кросплатформенним та надасть актуальну інформацію закладу, з можливістю виконувати онлайн основні операції майданчику;
 - Розробити концепцію стартап-проекту для можливості його комерціалізації;
 - Запропонувати варіанти подальшого покращення чи розширення системи.

2. ЗАСОБИ ЕФЕКТИВНОЇ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ

Для розроблення клієнтської частини інформаційної системи відповідно до вимог: отримання актуальної інформації та виконання базових операцій майданчику онлайн, кросплатформеність та швидкодія, було вирішено створити клієнтську частину як веб-застосунок, що вирішить проблему кросплатформеності та швидкодії, а для можливості виконання операцій онлайн та забезпечення актуальної інформації, веб-застосунок комунікуватиме з сервером.

Для створення веб-застосунку обрано мови програмування JavaScript та TypeScript, як покращення мови програмування JavaScript з додаванням типізації, фреймворк Angular, який дозволяє писати великі за архітектурою застосунки, які можуть легко розширюватись, NPM(Node Package Manager) менеджер пакетів, який дозволяє встановлювати та використовувати необхідні пакети, при створенні додатку, збірник проектів Webpack, який перетворює модулі проекту в формат зрозумілий браузеру.

Також було використано систему контролю версій Git для контролю операцій над проектом та збереження копії на віддаленому репозиторії на GitHub.

2.1. Система контролю версій Git

Система контролю версій (Version Control System, VCS) — це система, що записує зміни в файл або набір файлів і дозволяє повернутися пізніше до певної версії.

За своїм базовим можливостям Git схожий з Mercurial (і іншими VCS), але завдяки ряду переваг (висока швидкість роботи, можливість інтеграції з іншими VCS, зручний інтерфейс) Git вийшов в лідери ринку розподілених систем контролю версій[3]. Тому для даного проекту було також вибрано систему контролю версій Git.

Система контролю версій Git дозволяє повернути файли до стану, в якому вони були до змін, повернути проект до вихідного стану, побачити зміни, побачити, хто виконував зміни чи викликав проблему і багато іншого.

Використання СКВ також означає в цілому, що, якщо щось зламалось або втратись файли, ці проблеми можуть бути з легкістю виправлені[4].

Отже, репозиторієм проекту є сховище на GitHub. При цьому завантажуючи проект через команду `git clone` ми отримуємо локальну копію проекту з якою можемо працювати.

Наші файли можуть бути в такому стані: `working directory`, `staging area`, `git repository`. Коли ми працюємо з файлами, додаємо нові, змінюємо існуючі, видаляємо, дані зміни знаходяться в `working directory` та для того аби перевести їх в `staging area` необхідно виконати команду `git add <.*>`[5]. Після того як наші зміни знаходяться в `staging area` вони переводяться в стан `git repository` командою `git commit -m <message>`. Ця команда фіксує зміни в нашому локальному репозиторії. Тепер щоб додати їх в віддалений репозиторій, тобто синхронізувати наш локальний з віддаленим репозиторієм, необхідно виконати команду `git push`[6].

При створенні даного застосунку також було використано `feature branching strategy`. Відповідно до даної стратегії, є базова гілка `master` в якій є повністю робоча версія застосунку. При розробленні нових функцій – `feature`, ми створюємо нову гілку, в якій повністю розроблюємо та тестуємо дану функціональність. Після успішної реалізації та проходження тестування гілка змержується в головну гілку `master`.

Приклад гілок проекту зображено на рисунку 2.1 .

Гілки створювались відповідно до обраної стратегії, а значить відповідно з розроблюваними функціями. Тому було створено гілки:

- `feature-personal-info` — для розробки особистого кабінету та сторінки редагування інформації користувача;
- `feature-schedule` — розробка сторінки розкладу майданчику;
- `feature-shared` — сторінка з загальними компонентами, які будуть необхідні в різних частинах додатку;

- feature-authorization — створення сторінки авторизації та реєстрації користувача;
- feature-saving-personal-info — оновлення особистої інформації користувача;
- feature-tickets — створення сторінки з квитками та додавання функціонування для повернення квитків.

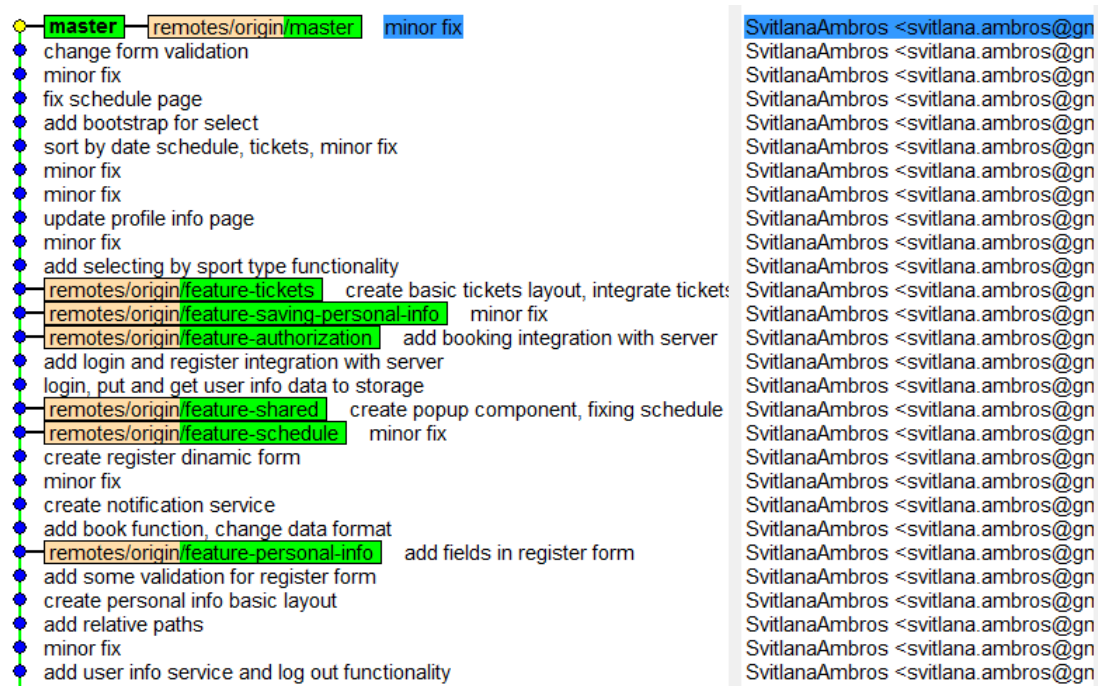


Рисунок 2.1 — Feature branching strategy

Отже, в випадку додавання нових функцій для додатку необхідно створити нову гілку та перейти на неї з головної гілки, з використанням команди:

```
git checkout -b <назва гілки>
```

Після закінчення роботи гілку необхідно об'єднати з головною гілкою використовуючи команду:

```
git merge
```

Дана стратегія розбиття гілок досить зручна, оскільки ми зберігаємо робочу версію продукту, а після додавання нового функціоналу та успішного проходження тестування оновлюємо існуючу версію, надаючи інкремент продукту.

2.2. Менеджер пакетів NPM

NPM (Node Package Manager) — це менеджер пакунків для роботи з JavaScript.

NPM було використано для завантаження пакетів для роботи з JavaScript, TypeScript та Angular, а також для запуску проекту локально.

Приклад команди для завантаження пакетів глобально на комп'ютер:

```
npm install -g @angular/cli
```

Для завантаження та додавання бібліотеки в існуючий проект використовується також дана команда з флагом `--save-dev`, яка додає дану залежність в `package.json`[7].

Після завантаження проекту з репозиторію використовується команда `npm install`, яка завантажує залежності проекту, які необхідні для його роботи та прописані в залежностях файлу `package.json`. Після успішного завантаження залежностей проекту, його можна запустити використовуючи команду `npm run start`, яка запускає додаток на `localhost:4200`.

Інші існуючі команди прописані також в `package.json` файлі та можуть доповнюватись. При цьому прописують назву команди та її опис, для її запуску потім використовується `npm run <назва створеної команди>`, що може бути зручним у випадку якщо ми додаємо безліч флагів та дозволяє не дублювати великі команди вручну в командному рядку[8].

2.3. Мова програмування TypeScript

TypeScript — мова програмування, як засіб розробки веб-додатків, що розширює JavaScript.

Для створення веб-застосунків зазвичай використовується мова програмування JavaScript, але вона не є зручною в випадку великих проектів[9]. Оскільки мова не має типізації важко розуміти, яким типом даних є яка змінна. Особливо це стосується ситуацій, коли ми працюємо з об'єктами. Наприклад, якщо з

серверу приходить великий json, з використанням JavaScript важко визначити шлях для отримання конкретного поля[10]. А якщо таких об'єктів буде багато, то пам'ятати всі типи об'єктів буде неможливим.

Тому, для вирішення цієї проблеми було обрано використовувати мову програмування TypeScript, адже вона є мовою сумісною з JavaScript і компілюється в JavaScript[11]. А це означає, що після компіляції, програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверної платформою Node.js[12].

TypeScript відрізняється від JavaScript можливістю явного статичного призначення типів, підтримкою класів, як в об'єктно-орієнтованих мовах. Крім того, TypeScript покращує читабельність і повторне використання коду, а також підтримку модулів. З використанням TypeScript помилки відловлюються в процесі розробки та компіляції, а також проект стає простішим для розширення.

2.4. Angular фреймворк

Для створення веб-застосунку можна використовувати мову JavaScript, але вона не є зручною для створення великих застосунків. Наприклад, для відловлювання подій треба визначити елемент, додати на нього функцію і лише після того можна написати, що саме ми бажаємо роботи в випадку даної події. Написання таких речей є повторюваним процесом та забирає багато часу. Замість того, щоб приділити більше часу на правильність бажаної обробки події ми витрачаємо її на більш низькорівневі речі.

Тому, було вирішено використати фреймворк — Angular, який надає можливість будувати інтерактивні і динамічні веб-додатки та значно спрощує вирішення наведених вище проблем .

Angular є структурною основою для динамічних веб-додатків, яка дозволяє використовувати HTML в якості мови шаблону, а потім розширити синтаксис HTML для вираження компонентів програми. За допомогою прив'язки даних і

впровадження залежностей можна виключити більшу частину коду, який довелося б писати.

Особливості, які вплинули на вибір даного фреймворку:

- В якості мови шаблону використовується Typescript;
- Angular націлений на розробку односторінкових додатків, тобто SPA-рішень (Single Page Application)[13]. Застосунок відкритого спортивного майданчику також є SPA додатком, а це означає, що запити на сервер та отримання результатів відбувається без перенаправлення на нову сторінку[14];
- Angular надає клієнтську MVC-інфраструктуру, яка допомагає у запуску і створенню динамічних додатків на сучасному рівня якості[15]. Це надало можливість відділяти представлення від бізнес логіки;
- Програми, написані на Angular, сумісні з різними браузерами. Оскільки, Angular перетворює код з TypeScript в JavaScript, який може бути прочитаний браузером[16];
- Проста маршрутизація з використанням Router;
- Структура Angular розширює синтаксису HTML і легко створює повторно використовувані компоненти по директивам;
- Надає можливість використовувати препроцесори стилів, наприклад, Sass.

Отже, Angular — це фреймворк для створення великомасштабних, високопродуктивних і простих в обслуговуванні веб-додатків. Швидке відображення сторінок значно покращує сприйняття веб-додатків, написаних в рамках Angular.

2.5. Постачальник модулів Webpack

При написанні додатків на TypeScript та Angular додаток складається з великої кількості модулів. Веб-браузери не розуміють як інтерпретувати ці модулі. Потрібно або додати весь JavaScript-код в один файл і імпортувати його, або потрібно додати всі файли вручну на сторінку за допомогою тега script.

Для вирішення цієї проблеми було обрано бандлер модулів Webpack (module bundler), який комбінує різні модулі та їх залежності в один файл в правильному порядку та перетворює і комбінує код в формат, зрозумілий браузеру(рисунок 2.2)[17].

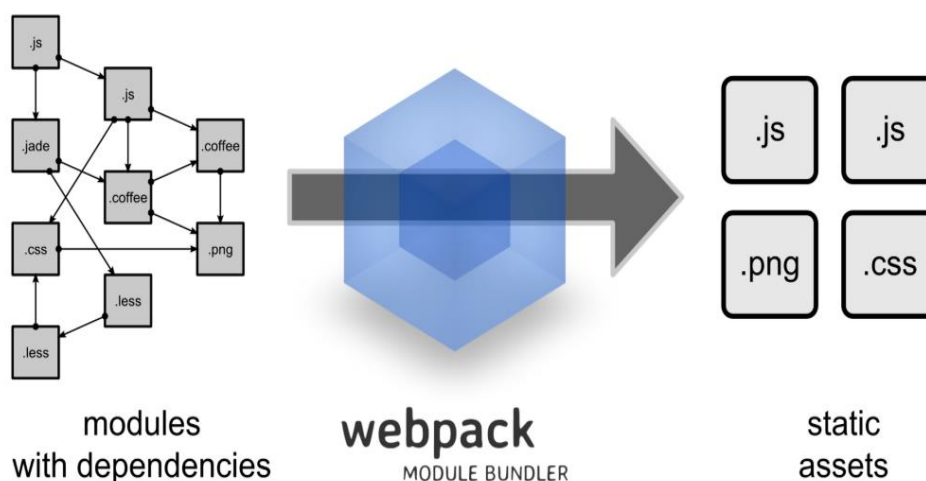


Рисунок 2.2 — Webpack

Отже, Webpack виконує такі перетворення з нашим додатком:

- Переведення файлів з es6 в es5;
- Переведення файлів .scss файли в звичайний .css;
- Мініфікація файлів (зменшує розмір файлу);
- Видалення коментарів з усіх файлів (для подальшого зменшення розміру файлу);
- Додавання всіх призначених шрифтів та іконок в вихідний файл (щоб вони були доступні відразу, без затримки, пов'язаної з додатковими HTTP запитами);
- Збір .css збірки (bundles) в один файл;
- Автоматично додає файли як теги `<link>` або `<script>` в ваш .html файл (не потрібно вручну змінювати теги кожен раз, коли змінюються версії файлів).

Таки чином, після збірки проекту отримуємо єдиний js, css, html файл.

Також, при використанні Webpack для створення додатка Angular створюється package.json файл, який виглядає як на рисунку 2.3.

```
{
  "name": "playground-web-app",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build --prod",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^5.1.0",
    "@angular/common": "^5.1.0",
    "@angular/compiler": "^5.1.0",
    "@angular/core": "^5.1.0",
    "@angular/forms": "^5.1.0",
    "@angular/http": "^5.1.0",
    "@angular/platform-browser": "^5.1.0",
    "@angular/platform-browser-dynamic": "^5.1.0",
    "@angular/router": "^5.1.0",
    "angular-font-awesome": "^3.1.2",
    "angular-webstorage-service": "^1.0.2",
    "bootstrap": "^4.3.1",
    "bootstrap-select": "^1.13.12",
    "core-js": "^2.4.1",
    "font-awesome": "^4.7.0",

```

Рисунок 2.3 — Файл package.json

І також додається в папку проекту для управління конфігурацією typescript файл tsconfig.json (рисунок 2.4).

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2017",
      "dom"
    ],
    "baseUrl": "src",
    "paths": {
      "@app/*": [
        "app/*"
      ],
      "@core/*": [
        "app/core/*"
      ],
      "@shared/*": [
        "app/shared/*"
      ],
      "@authorization/*": [

```

Рисунок 2.4 — Файл tsconfig.json

Файл package.json містить секцію devDependencies та визначення пакетів typescript, webpack і webpack-dev-server, які будуть потрібні при розробці, а

також купу пакетів, які призначені, головним чином, для завантаження тих чи інших типів файлів.

У секції `scripts` визначені команди. Перша команда “`start`” буде запускати `webpack-dev-server`, який в свою чергу буде запускати додаток. Друга команда “`build`” буде потрібно для компіляції всіх необхідних збірок з розрізнених модулів.

Файл `tsconfig.json` містить такі параметри як “`baseUrl`”, “`paths`”(шляхи, які необхідні будуть для відносних імпортів), “`outDir`” (директорія, де будуть знаходитись вихідні файли, які були створені за допомогою Webpack, після перетворення модулів проекту).

2.6. Препроцесор стилів Sass

Написання CSS інколи призводить до того, що таблиця стилів стає величезною і тому складною обслуговуваною. І ось в такому випадку допомагає препроцесор. Sass дозволяє використовувати функції недоступні в самому CSS, наприклад, змінні, вкладеності, міксини, успадкування та інші речі, які повертають зручність написання CSS[18].

Як тільки починає використовуватися Sass, препроцесор обробляє Sass-файл і зберігає його як простий CSS-файл, який може бути використаний на будь-якому сайті.

Найпростіший спосіб отримати такий результат — використовувати термінал. Після того, як Sass встановлений, ви можете компілювати ваш Sass в CSS, використовуючи команду `Sass`. Також можна вказати папки для відстеження змін і куди зберігати компілюють CSS файли, для цього достатньо вказати шляхи і розділити їх двокрапкою[19]. В проекті перетворенні Sass в CSS виконує webpack.

Отже, змінні — спосіб зберігання інформації, яку можна використовувати протягом написання всіх стилів проекту. Можна зберігати в змінних кольори, стеки шрифтів або будь-які інші значення CSS, які будуть використовуватися. Щоб створити змінну в Sass потрібно використовувати символ `$`[20].

Приклад використання змінних зображено на рисунку 2.5, де створюються змінні з основними кольорами, які будуть використані в додатку. Це надає нам переваги, якщо у випадку зміни кольору, його необхідно змінити лише в одному файлі, а не всюди, де був би прописаний цей колір в випадку використання стилів в звичайному CSS файлі.

```
$dark-grey-color: #292525;
$white-color: #ffffff;
$black-color: #000000;

$primary-color: #009688;
$light-primary-color: #87cfc9;
$dark-primary-color: #00796b;

$accent-color: #ffc107;
$dark-accent-color: #e3ac06;

$light-grey-color: #f2f2f2;
$grey-color: #bdbdbd;
$middle-grey-color: #767373;
$dark-grey-color: #3a3636;

$book-btn-color: #0d930d;
$book-btn-active-color: #077207;

$error-color: #d64c4c;
$info-color: #e7c933;
$success-color: #229154;
```


Рисунок 2.5— Файл tsconfig.json

При написанні HTML, помітно, що він має чітку вкладену і візуальну ієрархію. З CSS це не так.

Sass дозволяє вкладати CSS селектори таким же чином, як і в візуальній ієрархії HTML[21]. Але тут треба пам'ятати, що надмірна кількість вкладень робить ваш документ менш читабельним, що вважається поганою практикою, а нормальна кількість вкладень покращує вигляд стилів та робить їх простіше для читання та швидкого знаходження потрібної частини коду.

Прикладом ієрархії та вкладеності стилів зображено на рисунку 2.6. Використання такої вкладеності значно скорочує код, відмінняє дублювання, та робить код більш читабельним.

CSS має можливість імпорту, яка дозволяє розділити CSS-файл на більш дрібні і полегшити `@ import`, але в CSS файлах для цього створюється ще один HTTP-запит. Sass бере ідею імпорту файлів через директиву `@import`, але замість створення окремого HTTP-запиту Sass імпортує вказаний в директиві файл в той, де він викликається.



```

@import '../shared/color.constants.scss';

.schedule {
  padding: 30px;
  background: $white-color;
  box-shadow: 10px 10px 10px $grey-color;
  min-height: 75vh;

  &-wrap {
    padding: 10px 30px;
    background: $light-grey-color;
  }

  &-row {
    display: flex;
    flex-wrap: wrap;
  }

  &-col {
    margin: 0 14px;
  }
}

.filter {
  margin: 20px 20px 25px;

  &__select {
    width: 200px;
  }
}

```

Рисунок 2.6— Стили в Sass файлі

Тобто на виході виходить один CSS-файл, скомпільований з декількох фрагментів. Приклад імпорту також зображено на рисунку 2.6, де ми імпортуємо змінні з константами, для подальшого їх використання в даному файлі стилів.

2.7. Реактивне програмування RxJS

Реактивне програмування — програмування з асинхронними потоками даних.

Реактивний підхід підвищує рівень абстракції коду надає можливість сконцентруватися на взаємозв'язку подій, які визначають бізнес-логіку, замість того, щоб постійно підтримувати код з великою кількістю деталей реалізації. Крім того

код з використанням реактивного програмування коротший та простіший для читання.

При написанні веб-додатків виникає проблема обробки різноманітних UI-подій. В даній ситуації додавання реактивного програмування може покращити та полегшити написання коду. Оскільки, натискання кнопки може спричинити відправку запиту на сервер, отримання розкладу, фільтрацію результату за видами спорту та виведення користувачеві з можливістю бронювання конкретної події та інше. Визначення необхідної послідовності дій в певному порядку та після отримання результатів з серверу було б складно відловлювати без реактивного програмування. Крім того, без асинхронності, дані події можуть блокувати UI. Тому було використано бібліотека RxJS для роботи з асинхронними і заснованими на подіях програмами з використанням спостережуваних послідовностей[22].

Бібліотека надає основний тип Observable, кілька допоміжних типів (Observer, Schedulers, Subjects) і оператори роботи з подіями як з колекціями (map, filter, reduce, every і подібні з JavaScript Array)[23].

Observable – спостережуваний, ітеруємий об’єкт, відповідно до шаблону програмування “Спостерігач”, об’єкт за яким можна спостерігати і якимось реагувати, якщо він змінюється. Observable є множинним Promise. Функції, які ми визначаємо коли підписуємось на Observable, називаються спостерігачами та записуються через subscribe (рисунк 2.7)[24].

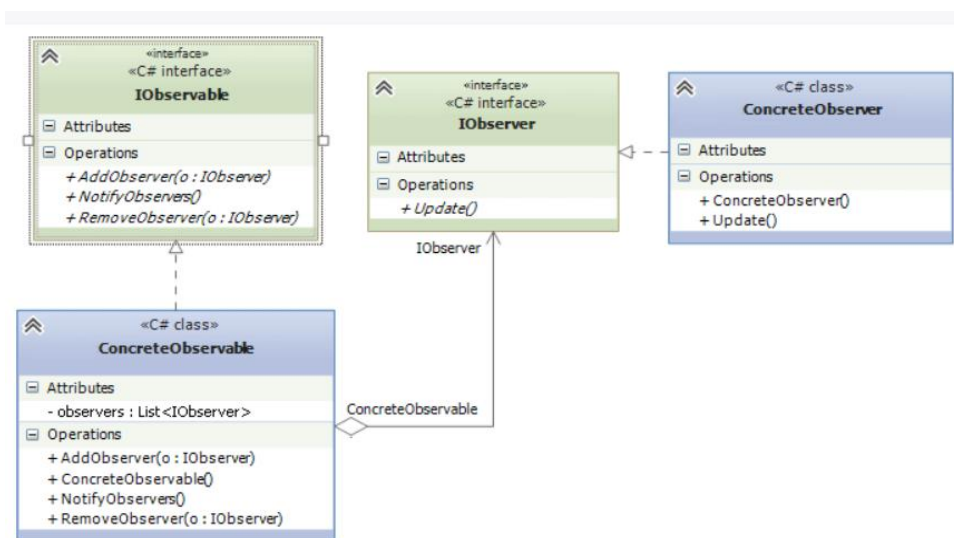


Рисунок 2.7— Патерн “Спостерігач”

Функція підписки містить 3 параметри, функцію, яка викликається коли буде отримано результат, наступна функція для обробки помилок та 3 параметр функція для обробки завершення.

Приклад реактивного програмування в додатку, зображено на рисунку 2.8. HttpClient, який використовується для запиту на сервер, повертає об'єкт Observable. Відповідно до цього, можна підписатись на його зміни.

Методи map, filter, sort, схожі на методи масиву дозволяють перетворювати потік після отримання результатів після чого повертають його.

```
constructor(private http: HttpClient) { }

public getScheduleForPeriod(sportType: string): Observable<ScheduleDay[]> {
    if (sportType !== 'All') {
        return this.http.get<ScheduleDay[]>(GET_SCHEDULE_FOR_PERIOD).pipe(
            map((data: ScheduleDay[]) => {
                return data.map((item: ScheduleDay) => {
                    return { ...item, dateUI: this.getDateFromString(item.date) };
                })
            })
            .sort((a: ScheduleDay, b: ScheduleDay) => a.dateUI.getTime() - b.dateUI.getTime())
            .map((day: ScheduleDay) => {
                return {
                    ...day,
                    events: day.events
                        .filter((event: ScheduleEvent) => event.sportType === sportType)
                };
            })
            .filter((day: ScheduleDay) => day.events.length > 0);
    }
}
```

Рисунок 2.8— Використання RxJS

В даному прикладі, HttpClient повертає Observable після отримання результату з серверу, потім потік перетворюється та фільтрується відповідно до обраного виду спорту та сортується за датами після чого повертається назад новий потік. Якщо не обрано вид спорту, то повертається весь отриманий з серверу розклад.

Таким чином з методу в нашому сервісі ми повертаємо Observable. Після чого через метод subscribe в компоненті, підписуємось та отримуємо вже відсортовані і перетворені дані з сервісу.

Додавання асинхронності дозволяє підписуватись на Observable об'єкт та не блокувати головний екран, що дозволить користувачеві надалі працювати з додатком. Коли будуть отримані результати спрацює callback функція та результат буде оброблений. Також в методі subscribe можна відловлювати помилки та нотифікувати користувача.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ВІДКРИТОГО СПОРТМАЙДАНЧИКУ

При створенні веб-додатку було визначено базову архітектуру, основними частинами якої є модулі, компоненти та сервіси. Для спілкування з сервером було використано шаблон архітектури REST.

3.1. Клієнт-серверна архітектура REST

Оскільки, однією з задач є надання актуальної інформації спортивного закладу користувачеві, було вирішено обрати клієнт-серверну архітектуру (рисунок 4.1), адже таким чином на сервері зберігатиметься стан ресурсу, а клієнти підключенням до серверу отримуватимуть та зберігатимуть в себе стан додатку.

Для спілкування між клієнтом та сервером було розглянуто протокол та стандарт SOAP, а також архітектуру в стилі REST. Проаналізувавши їх було обрано архітектуру REST, оскільки дана архітектура не залежить від мови та платформи, базується на надійному протоколі передачі даних HTTP та повністю використовує всі його можливості, в той час як SOAP було створено з розрахунку на використання відразу декількох протоколів передачі даних, в тому числі HTTP, в результаті чого задіяно лише невелику частину можливостей цього протоколу[25]. Також, REST базується на передачі стану, в той час як SOAP на передачі дії, крім того, SOAP базується на XML форматі, що впливає на швидкість обробки запитів, і часом для вирішення даної проблеми, параметри відправляються напряму в HTTP запитах, що більш відповідає архітектурі стилю REST. Архітектура стилю REST надає також набір CRUD(create, read, update, delete) методів для обробки об'єктів, що є досить зручним, адже формує набір чітко визначених існуючих операцій[26].

У архітектурі стилю REST кожна інформаційна одиниця вважається ресурсом, і ці ресурси адресуються за допомогою універсальних ідентифікаторів

(URI). Як правило, ідентифікатори представляють собою веб-посилання. Дії над ресурсами виконуються при використанні обмеженого набору чітко визначених операцій. REST як стиль клієнт-серверної архітектури визначає конструкцію для обміну станом цих ресурсів із застосуванням певного інтерфейсу і протоколу.

Для розробки веб-служби з передачею стану, необхідно розуміти деталі протоколу передачі гіпертексту (HTTP) і унікальних ідентифікаторів ресурсів (URI), а також дотримуватись певних принципів проектування. По суті, це означає, що кожен URI є поданням певного об'єкта. З даним об'єктом можна взаємодіяти за допомогою запитів HTTP ²⁷:

- GET — для одержання змісту;
- DELETE — для видалення вмісту;
- POST — для створення вмісту;
- PATCH – часткове оновлення, зміна;
- PUT — для оновлення вмісту.

В даному додатку було використано основні методи GET для отримання інформації, POST для її відправки та PUT для оновлення інформації.

Методи серверу для роботи з ресурсами було відображено на Swagger документації та надано для інтеграції клієнтської частини з серверною частиною(рисунок 3.1).

GET	/v1/events
POST	/v1/events
POST	/v1/events/book
POST	/v1/events/check
POST	/v1/events/unbook
GET	/v1/events/{eventId}
PUT	/v1/users
OPTIONS	/v1/users
POST	/v1/users/login
OPTIONS	/v1/users/login
POST	/v1/users/register

Рисунок 3.1 — Приклад документації серверних методів

Наприклад, метод POST login, registration використовується для відправки даних користувача. Оскільки надсилається особиста інформація використовується саме метод POST, а не GET, адже POST відправляє параметри в body, що робить їх прихованими, в той час як GET додає параметри в рядок запиту, тим самим дані стають відкритими. Для отримання розкладу використовується метод GET events. Для оновлення особистих даних користувача використовується метод PUT users.

На діаграмі послідовностей (рисунок 3.2) відображено приклад спілкування між клієнтом та сервером.

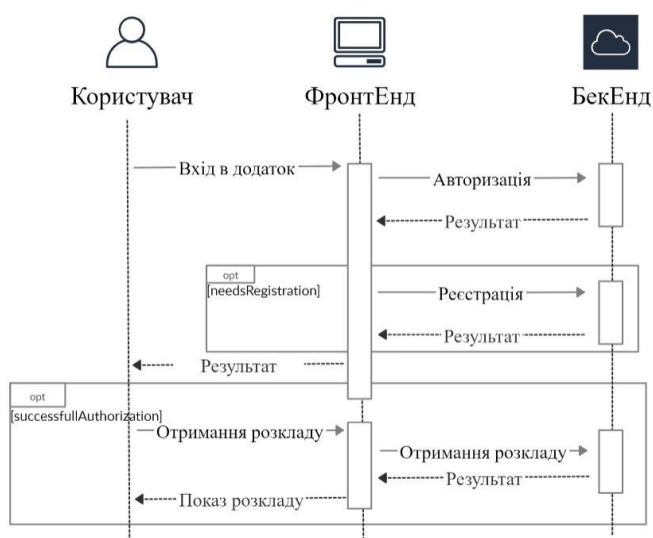


Рисунок 3.2 — Діаграма послідовностей

Отже, після введення та підтвердження входу чи реєстрації, додаток надсилає дані з використанням серверного метода POST login чи registration, після чого отримує результат входу з серверу. В випадку успішної операції входу, користувач потрапляє в особистий кабінет, де за замовчуванням відкривається вкладка з розкладом, а відповідно відправляється запит на сервер для отримання розкладу GET events. В випадку неуспішного входу, користувачеві необхідно повторити введення особистих даних. За таким самим принципом відбувається виклик методів серверу для отримання квитків користувача, бронювання події чи скасування бронювання події, а також методу оновлення особистої інформації користувача.

3.2. Архітектура веб-додатку

Основними складовими Angular додатку є модулі, компоненти та сервіси (рисунок 3.3). Відповідно до цього додаток було розбито на основні логічні одиниці – модулі, а саме : Core(містить основні постійні складові частини додатку, які є singleton), Shared(містить елементи, які використовуються в різних частинах додатку, тобто в різних модулях), Schedule(модуль з графіком занять та деталями), PersonalInfo(модуль з інформацією користувача), Authorization(модуль, що забезпечує авторизацію та реєстрацію), Tickets(модуль з актуальними квитками).

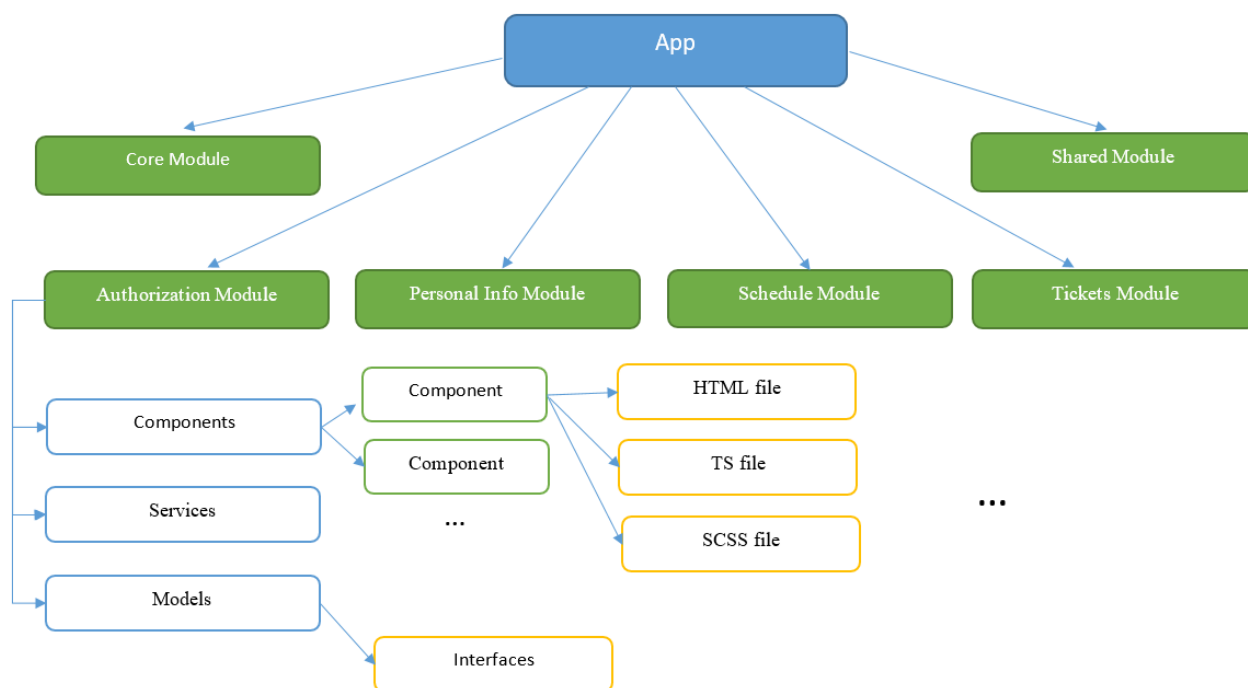


Рисунок 3.3 — Архітектура веб-додатку

Кожен модуль було розділено на наступні частини: компоненти, сервіси та модель. Модель містить інтерфейси, які визначають структуру даних. Сервіси виконують бізнес логіку додатку, наприклад, завантаження та обробка даних. Компоненти є графічним представленням, яке відображається на екрані. Компоненти складаються з файлу розмітки, файлу стилів, та файлу ts який дозволяє

додавати певну логіку на наше представлення, наприклад підключення до сервісу, який завантажує певні дані.

Відповідно до схеми архітектури зображеної на рисунку 3.3 можна побачити розбиття проекту на класи, які зображено на діаграмі класів(рисунок 3.4). На діаграмі зображено часткову діаграму класів з головними модулями та детальним розглядом одного з модулів, інші модулі побудовані за таким самим принципом.

Основною і базовою складовою Angular проектів є `app.component.ts`, `app.module.ts`, де `app.module.ts` є стартовим модулем, а `app.component.ts` – базовим компонентом, який завантажує в свою чергу наступні компоненти. Даний модуль використовується в Router. Router модуль додає навігацію по проекту, відповідно до визначених переходів та включає інші основні модулі Core, Shared, Schedule, PersonalInfo, Authorization та відповідно може навігувати користувача до них.

В даній діаграмі класів зображено більш детальне розбиття Authorization модуля, який складається з наступних базових компонентів(представлення) — `authorization` компонент, який включає в себе `register` та `login` компонент, та відображає конкретний в залежності від вибору реєстрації чи входу користувача в систему.

В модулі знаходиться сервіс `authorization.service`, який зв'язує компоненти з сервером та виконує основні запити на сервер для авторизації чи реєстрації користувача. Крім того у випадку успішного входу користувача в систему сервіс зберігає особисті дані користувача з використанням `webstorage-service` в `localStorage`, що дозволяє зберегти дані в систему, якщо користувач не виконував вихід з системи. Сервіс надсилає результат входу користувача в систему компоненту. В залежності від результату, компонент або викликає роутер для навігації та відкриття сторінки особистого кабінету користувача, або залишає користувача на сторінці авторизації чи реєстрації для повторного введення даних.

Модель, що описує формат зберігання інформації користувача, описана в інтерфейсі `user-info.model` та використовується в компонентах та сервісі даного модуля.

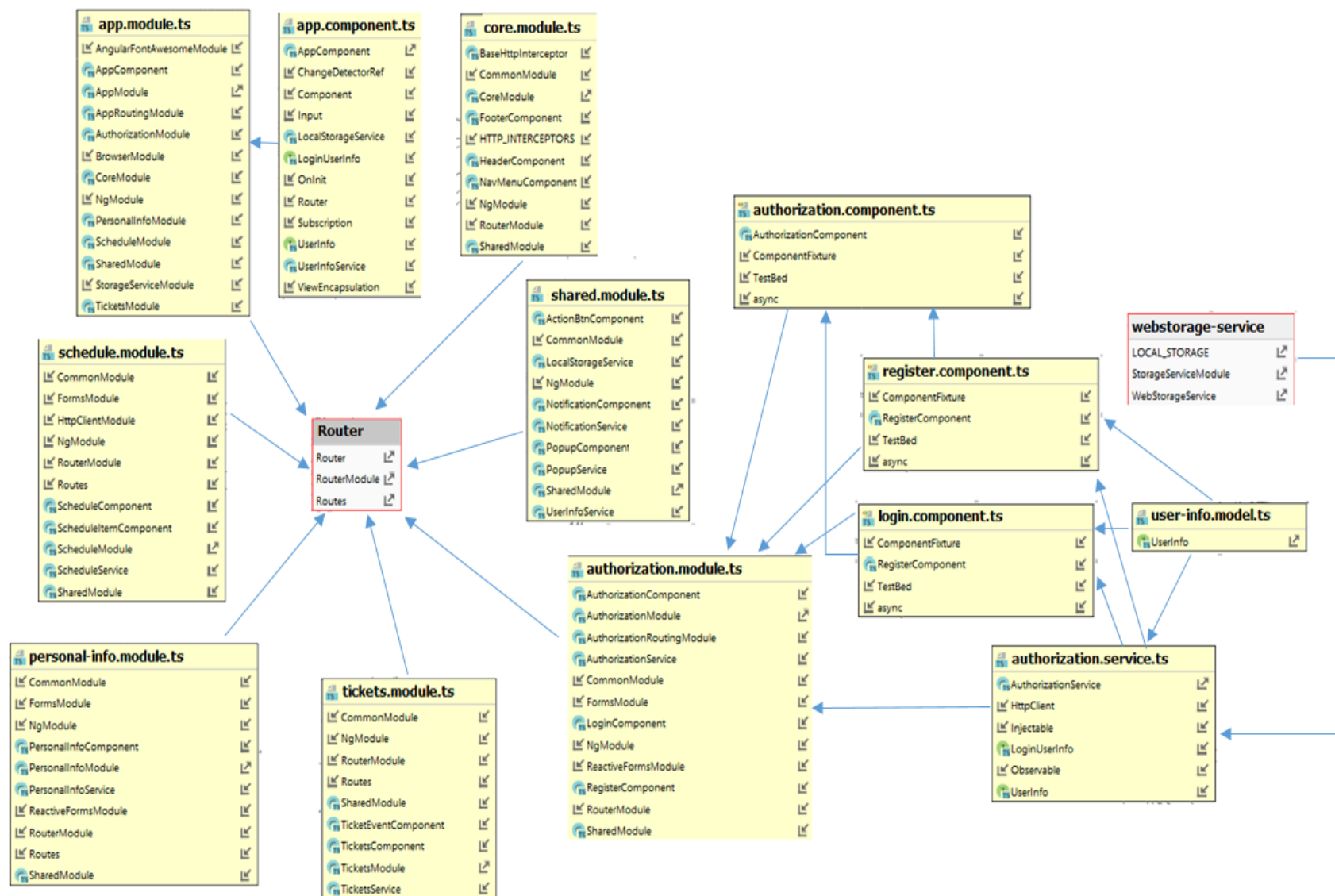


Рисунок 3.4 — Діаграма класів веб-додатку

Інші модулі побудовані таким самим чином, тобто мають компоненти, які визначають відповідну розмітку, яка характеризує даний модуль, сервіси, де описана бізнес логіка, та модель, яка визначає інтерфейси необхідних нам форматів даних.

3.3. Модулі веб-додатку

Angular додатки є модульними, а Angular має власну модульну систему під назвою NgModules. NgModules є контейнерами для згуртованого блоку коду, присвяченого домену програми, робочому процесу або тісно пов'язаному набору можливостей. Вони можуть містити компоненти, сервіси та інші файли коду, обсяг яких визначений NgModule. Вони можуть імпортувати функціонал, який експортується з інших NgModules, та експортувати вибрану функціональність для використання іншими NgModules.

Кожен додаток Angular має щонайменше один клас NgModule, кореневий модуль, який умовно називається AppModule і знаходиться у файлі з назвою app.module.ts. Коли запускається додаток, завантажується кореневий NgModule[28].

Створюваний додаток має багато функціональних модулів, які були відображені на схемі архітектури додатку, а саме core, shared, tickets, personal-info, authorization.

Корінь NgModule для програми названий так, тому що він може включати дочірні NgModules в ієрархію будь-якої глибини.

Метадані NgModule в NgModule визначаються класом з декоратором @NgModule (). Декоратор @NgModule () — це функція, яка приймає один об'єкт метаданих, властивості якого описують модуль (рисунок 3.5).

Найважливіші властивості такі:

- declarations: компоненти, директиви та пайпи, що належать до цього модулю;

- `exports`: підмножина декларацій, які мають бути видимими та корисними у шаблонах компонентів з інших `NgModules`, тобто з інших модулів;
- `imports`: інші модулі, експортовані класи які потрібні шаблонам компонентів, оголошеним у цьому `NgModule`;
- `providers`: сервіси послуг, які цей `NgModule` додає глобальній колекції сервісів; вони стають доступними у всіх частинах програми. (Також можна вказати провайдерів на рівні компонентів);
- `bootstrap`: основний вигляд програми, який називається кореневим компонентом, який розміщує всі інші представлення програм. Тільки кореневий `NgModule` повинен встановлювати властивість завантажувальної програми [29].

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';

import { SharedModule } from '@shared/shared.module';
import { TicketsComponent } from '@tickets/tickets/tickets.component';
import { TicketsService } from '@tickets/services/tickets.service';
import { TicketEventComponent } from '@tickets/ticket-event/ticket-event.component';

const routes: Routes = [
  {
    path: 'tickets',
    component: TicketsComponent
  }
];

@NgModule({
  imports: [
    CommonModule,
    SharedModule,
    RouterModule.forRoot(routes)
  ],
  declarations: [
    TicketsComponent,
    TicketEventComponent
  ],
  providers: [
    TicketsService
  ],
  exports: [
    TicketsComponent,
    RouterModule
  ]
})
```

Рисунок 3.5 — Базова структура модулю

Розглянувши приклад даного модулю квітків(рисунок 3.5), можна побачити, що для його роботи необхідно імпортувати `commonmodule` — загальний модуль ангуляра, `sharedmodule` — модуль з спільними частинами проекту, які необхідні в даному модулі, `routermodule` — для можливості навігації. Крім того, в даному модулі декларуються `ticketscomponents`, `ticketeventcomponent`, які є компонентами

всередині даного модулю. При чому експортується лише `ticketscomponents`, а це значить що імпортувавши даний модуль, можна буде звернутись лише до `ticketscomponents`, а `ticketeventcomponent` буде невидимим, оскільки він інкапсульований всередині іншого модуля. В провайдері додається сервіс `ticketsservice`, який виконує запити на сервер для отримання квитків.

3.4. Сервіси веб-додатку

Сервіс — це широка категорія, що включає будь-яке значення, функцію чи покращення, яка потрібне додатку. Сервіс це зазвичай клас із вузькою, чітко визначеною метою. Він повинен робити щось конкретне і робити це добре.

Angular відрізняє компоненти від сервісів для підвищення модульності та повторного використання. Відокремлюючи функціональний вигляд компонента від інших видів обробки, ви можете зробити класи компонентів гнучкими та ефективними.

В ідеалі завдання компонента полягає в тому, щоб надати представлення більше нічого. Компонент повинен представляти властивості та методи прив'язки даних для того, щоб опосередковувати перегляд (наданий шаблоном) та логікою програми (яка часто включає деяке поняття моделі), яка необхідна для відображення.

Компонент може делегувати певні завдання сервісам, наприклад, отримання даних із сервера, перевірка введення користувача. Визначаючи такі завдання обробки в сервісі обслуговування, ці завдання стають доступними для будь-якого компонента.

Angular допомагає слідувати принципам, полегшуючи розподіл логіки програми на сервіси та робить ці сервіси доступними для компонентів шляхом введення залежності.

На рисунку 3.6 зображено приклад сервісу, який виконує завантаження розкладу заходів майданчику з серверу.

Сервіс може використовувати інший сервіс, наприклад сервіс `HttpClient` для отримання даних асинхронно з сервера.

DI(Dependency injection) підключається до Angular та використовується скрізь для надання нових компонентів послугами чи іншими необхідними речами. Компоненти використовують сервіси; тобто ви можете ввести сервіс в компонент, надавши компоненту доступ до цього сервісу.

Щоб визначити клас як сервіс в Angular, використовуються декоратор `@Injectable()`, та щоб надати метадані, які дозволяють Angular вводити його в компонент як залежність.

```
@Injectable()
export class ScheduleService {

  constructor(private http: HttpClient) { }

  public getScheduleForPeriod(sportType: string): Observable<ScheduleDay[]> {
    if (sportType !== 'All') {
      return this.http.get<ScheduleDay[]>(GET_SCHEDULE_FOR_PERIOD).pipe(
        map((data: ScheduleDay[]) => {
          return data.map((item: ScheduleDay) => {
            return { ...item, dateUI: this.getDateFromString(item.date) };
          })
            .sort((a: ScheduleDay, b: ScheduleDay) => a.dateUI.getTime() - b.dateUI.getTime())
            .map((day: ScheduleDay) => {
              return {
                ...day,
                events: day.events
                  .filter((event: ScheduleEvent) => event.sportType === sportType)
              };
            })
            .filter((day: ScheduleDay) => day.events.length > 0);
        })
      );
    }
  }
}
```

Рисунок 3.6 — Сервіси

Аналогічно використовується декоратор `@Injectable()`, щоб вказати, що компонент або інший клас (наприклад, інший сервіс, пайп або `NgModule`) має залежність.

- Інжектор є основним механізмом. Angular створює інжектор для всіх додатків під час завантажувального процесу та додаткові інжектори за потреби;
- Інжектор створює залежності і підтримує контейнер випадків залежності, який він повторно використовує, якщо це можливо;
- Провайдер — це об'єкт, який повідомляє інжектору, як отримати або створити залежність.

Для будь-якої залежності, яка потрібна у додатку, необхідно зареєструвати провайдер з інжектором програми, щоб інжектор міг використовувати провайдера для створення нових екземплярів. Для сервісу провайдер, як правило, це і є сам клас сервісу.

Коли Angular створює новий екземпляр класу компонентів, він визначає, які сервіси чи інші залежності, необхідні компоненту, переглядаючи типи параметрів конструктора. Наприклад, конструктору `ScheduleItemComponent` потрібний `scheduleService`, `NotificationService`, `LocalStorageService`, `PopupService` (рисунк 3.7).

```
export class ScheduleItemComponent implements OnInit {
    public popupControls: PopupControls;
    public currentEvent: ScheduleEvent;

    public isActionPerformed = false;

    @Input() data: ScheduleDay;
    @Output() updateScheduleInfo = new EventEmitter();

    constructor(private scheduleService: ScheduleService,
                private notification: NotificationService,
                private localStorageService: LocalStorageService,
                private popupService: PopupService) { }

    ngOnInit() {
```

Рисунок 3.7 — Приклад DI

Коли Angular виявить, що компонент залежить від сервісу, він спочатку перевіряє, чи є у інжектора якісь наявні екземпляри цієї служби. Якщо запитуваний службовий екземпляр ще не існує, інжектор робить його за допомогою зареєстрованого провайдера та додає його до інжектора перед поверненням послуги в Angular.

Коли всі запитувані сервіси будуть визначені та повернені, Angular може викликати конструктор компонента з цими службами як аргументи.

Загалом необхідно зареєструвати хоча б одного провайдера будь-якої послуги, яку ви будете використовувати. Провайдер може бути частиною власних метаданих

сервісу, роблячи службу доступною всюди, або ви можете зареєструвати постачальників певних модулів або компонентів. Тобто необхідно зареєструвати провайдерів у метаданих сервісу (в декораторі `@Injectable ()`) або в метаданих `@NgModule ()` або `@Component ()`.

За замовчуванням команда Angular CLI `ng generating service` реєструє постачальника з кореневим інжектором для сервіса, включаючи метадані провайдера в декоратор `@Injectable ()`.

Коли сервіс додається на кореновому рівні, Angular створює єдиний спільний екземпляр Service та вводить його в будь-який клас, який його вимагає. Реєстрація провайдера в метаданих `@Injectable ()` також дозволяє Angular оптимізувати додаток, видаливши службу зі складеного додатка, якщо він не використовується.

Якщо реєструвати провайдера з певним NgModule, той самий єдиний екземпляр послуги доступний для всіх компонентів у цій NgModule. Щоб зареєструватися на цьому рівні, використовується властивість провайдеру декоратора `@NgModule ()` (рисунок 3.8). В розробленому веб-додатку, сервіси створювались на рівні модулів, тобто були єдиними для всіх компонентів даного модуля.

```
@NgModule({
  imports: [
    CommonModule,
    SharedModule,
    RouterModule.forRoot(routes),
    HttpClientModule,
    FormsModule
  ],
  providers: [
    ScheduleService
  ],
  declarations: [
    ScheduleComponent,
    ScheduleItemComponent
  ],
  exports: [
    ScheduleComponent,
    RouterModule
  ]
})
export class ScheduleModule { }
```

Рисунок 3.8 — Визначення сервісів в модулі

Якщо реєструвати постачальника на рівні компонента, створюється новий примірник сервісу з кожним новим екземпляром цього компонента. На рівні компонента сервіс реєструється через властивість провайдеру в метаданих `@Component ()`.

3.5. Компоненти веб-додатку

3.5.1. Компоненти та їх складові

Компонент контролює частину екрану, яка називається view.

Розмітка компонента включає в себе звичайну HTML розмітку та власне розмітку, яку ми створюємо самі, тобто наші компоненти, які створюють власні теги. Логіка програми компонента — це те, що він робить для підтримки перегляду всередині класу. Клас взаємодіє з представленням через API властивостей та методів.

Головний плюс, який надають компоненти полягає в тому, що ми з легкістю зв'язуємо поля класу з розміткою на екрані та контролюємо їх, на відміну від того як це відбувається в JavaScript.

Наприклад, `ScheduleComponent` має властивість `scheduleData`, яка містить масив даних розкладу (рисунок 3.9, 3.10).

```
import { Observable } from 'rxjs/Observable';

import { ScheduleDay } from '@schedule/models/schedule-day.model';
import { ScheduleService } from '@schedule/services/schedule.service';
import { SPORT_TYPES } from '@schedule/schedule/sport-types.constant';

@Component({
  selector: 'app-schedule',
  templateUrl: './schedule.component.html',
  styleUrls: ['./schedule.component.scss']
})
export class ScheduleComponent implements OnInit {
  public sportTypes;
  public currentSportType = 'All';
  public scheduleData: Observable<ScheduleDay[]>;
  constructor(private scheduleService: ScheduleService) { }

  ngOnInit() {
    this.loadScheduleInfo(this.currentSportType);
    this.sportTypes = SPORT_TYPES;
  }

  public loadScheduleInfo(type: string = this.currentSportType): void {
    this.scheduleData = this.scheduleService.getScheduleForPeriod(type);
  }

  public changeSportType(): void {
    this.loadScheduleInfo(this.currentSportType);
  }
}
```

Рисунок 3.9 — Файл .ts компоненту

Розмітка компоненту прив'язана до поля `scheduleData`, проходить циклом та виводить дані розкладу. Крім того, наш компонент реагує на зміну виду спорту.

```

<div class="schedule-wrap">
  <div class="container">
    <div class="schedule">
      <div class="title">
        Playground Schedule
      </div>
      <div>
        <div *ngIf="(scheduleData | async) as schedule; else loading">
          <div class="filter">
            <select class="filter__select custom-select custom-select-sm" [(ngModel)]="currentSp"
              (ngModelChange)="changeSportType()">
              <option *ngFor="let sport of sportTypes" class="select__option" [value]="sport">
            </option>
          </select>
        </div>
        <div class="schedule-row">
          <div class="schedule-col" *ngFor="let dayData of schedule">
            <app-schedule-item [data]="dayData" (updateScheduleInfo)="loadScheduleInfo()">
            </app-schedule-item>
          </div>
        </div>
      </div>
      <ng-template #loading>
        Schedule data is loading ...
      </ng-template>
    </div>
  </div>
</div>

```

Рисунок 3.10 — Файл .html компоненту

Після вибору виду спорту розмітка повідомляє компоненту, що відбулись зміни з фільтруванням даних та виклика необхідний метод.

Декоратор `@Component` ідентифікує клас безпосередньо під ним як клас компонентів та вказує його метадані. У наведеному прикладі коду(рисунок 3.9) можна побачити, що `ScheduleComponent` — це просто клас, у якому немає спеціальних позначень чи синтаксису. Це не компонент, поки його не позначать декоратором `@Component`.

Метадані для компонента вказують Angular, де отримати основні будівельні блоки, необхідні для створення та подання компонента.

Зокрема, він асоціює шаблон із компонентом або безпосередньо з вбудованим кодом, або за посиланням. Разом компонент та його шаблон описують представлення даних. У прикладі на рисунку 3.9 показано кілька найбільш корисних параметрів конфігурації `@Component` [30]:

— селектор: CSS-селектор, який повідомляє Angular створити та вставити екземпляр цього компонента, де б він не знайшов відповідний тег у шаблоні HTML.

Наприклад, якщо HTML програми містить `<app-schedule> </app-schedule>`, то Angular вставляє екземпляр перегляду `ScheduleComponent` між цими тегами;

- `templateUrl`: відносна адреса шаблону HTML цього компонента. Крім того, ви можете надати шаблон HTML в рядку як значення властивості шаблону. Цей шаблон визначає вигляд хоста компонента;
- провайдери: набір сервісів послуг, необхідних для компонента(може бути прописаний в модулі або в компоненті).

3.5.2. Data binding

З використанням Javascript без фреймворку нам необхідно було б відповідати за переміщення значень даних у елементи керування HTML та перетворення відповідей користувачів у дії та оновлення значень. Писання такої логіки натискання та додавання вручну є стомлювальним, схильним до помилок і важким для читання.

Angular підтримує двостороннє прив'язування даних, механізм узгодження частин шаблону з частинами компонента. Для цього додається прив'язки в розмітку до HTML шаблону, з частинами компонента, щоб сказати Angular, як з'єднати обидві сторони[31].

На наступній схемі показані чотири форми розмітки прив'язки даних(рисунок 3.11).

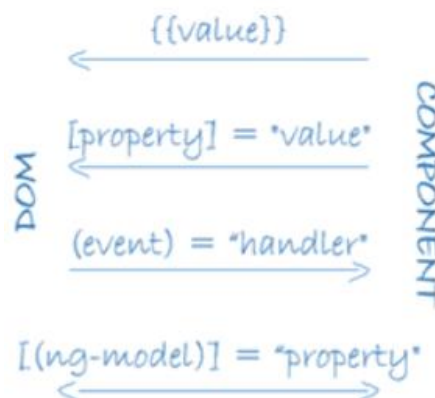


Рисунок 3.11 — Варіанти прив'язки до даних

Кожна форма має напрямок: до DOM, від DOM або обох (рисунок 3.11) та приклад використання в компонентах на рисунку 3.12.

- Інтерполяція `{{event.startTime}}` відображає значення властивості `event.startTime` компонента в елементі `<div>`;
- Прив'язка властивості `[isAvailable]` передає значення `emptyplaces` з батьківського `ScheduleComponent` у властивість дитини `AppActionBtnComponent`;
- Прив'язка події (`click`) викликає метод `bookEvent()` компонента, коли користувач натискає поле[32].

```
<div class="item__day">
  <div>
    | {{ data?.dateUI | date: 'd.MM.yyyy' }}
  </div>
  <div>
    | {{ data?.dateUI | date: 'EEEE' }}
  </div>
</div>
<div class="item-events">
  <div class="item-event" *ngFor="let event of data.events">
    <div class="item-event__title">
      <div class="item-event__time">
        | {{ event?.startTime }} - {{event?.endTime }}
      </div>
      <div class="item-event__type">
        | {{ event.sportType }}
        <span *ngIf="event?.emptyPlaceCount > 0": {{ event?.emptyPlaceCount }}</span>
      </div>
    </div>
    <app-action-btn [isAvailable]="event.emptyPlaceCount > 0" [availableMessage]="Book"
    | [noAvailableMessage]="No places" (clicked)="bookEvent(event)">
  </app-action-btn>

  <app-popup
    #popup
    *ngIf="popupControls.isOpened">
  </app-popup>
</div>
```

Рисунок 3.12 — Варіанти прив'язки до даних

Двостороння прив'язка даних (використовується переважно в керованих шаблонами формах) поєднує властивості та прив'язку подій в одній нотації.

Приклад двусторонньої прив'язки зображено на рисунку 3.13. `LoginComponent`, де використовується двостороннє зв'язування даних із директивою `ngModel`.

При двосторонній прив'язці значення властивості даних надходить у поле вводу з компонента, як і при прив'язці до властивості. Зміни користувача також повертаються до компонента, скидаючи властивість на останнє значення, як при прив'язці подій.

Таким чином ми прив'язуємо зміну `input` поля до зміну параметру `user.login`,

але зміна в нашому компоненті даного поля, теж спричинить оновлення поля в представленні(рисунок 3.13).

Angular обробляє всі прив'язки даних один раз для кожного циклу подій JavaScript, починаючи з кореня дерева компонентів програми через усі дочірні компоненти.

Прив'язка даних відіграє важливу роль у зв'язку між шаблоном та його компонентом, оскільки дозволяє динамічно оновлювати представлення компоненту в випадку зміни даних, до яких він прив'язаний. А також важлива для зв'язку між батьківським та дочірнім компонентами, адже дозволяє надсилати дані батьківських компонентів в дочірні та розподіляти логіку між компонентами.

```
<form [formGroup]="dynamicForm">
  <div class="form__row">
    <label for="login" class="form__label">Login ( Your Email )</label>
    <input type="text"
      id="login"
      class="form__value"
      formControlName="login"
      autocomplete="off"
      [(ngModel)]="user.login">
    <div *ngIf="isFormInvalid(dynamicForm, 'login')" class="error-message">
      <div *ngIf="dynamicForm?.controls['login']?.errors?.pattern">
        | Not right login format
      </div>
      <div *ngIf="dynamicForm?.controls['login']?.errors?.required">
        | * Login is obligatory field
      </div>
    </div>
  </div>
</form>
```

Рисунок 3.13 — Двустороння прив'язка до даних

Прив'язка подій у випадку батьківських та дочірніх компонентів дозволяє слухати події дочірніх компонентів. Адже відповідно до рекомендацій, щодо розподілення логіки між компонентами, існують розумні компоненти та компоненти, які відповідають за просто представлення. Таким чином, даний принцип дозволяє зосередити логіку в головних, тобто батьківських компонентах, а інші просто повідомлятимуть розумний компонент про те, що в дочірньому компоненті відбулись певні події. Але застосування даного принципу не завжди ефективне, тому необхідно дивитись по ситуації та цілі розбиття на компоненти в додатку.

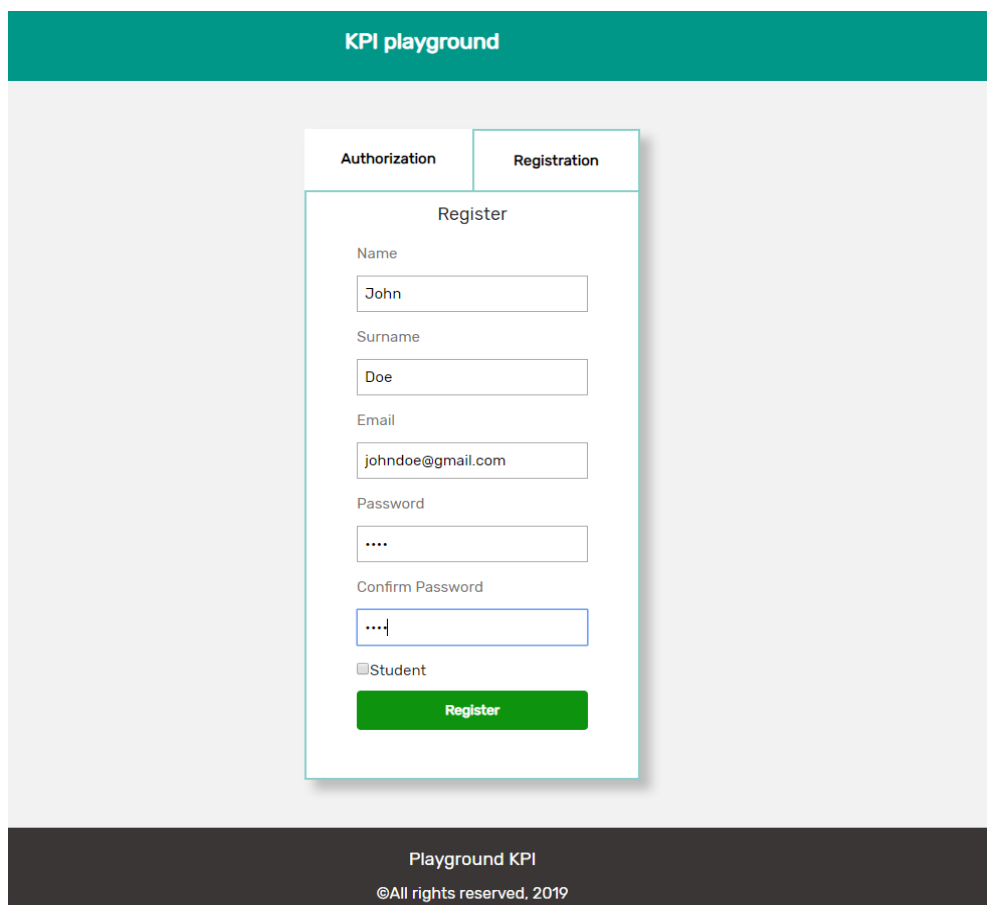
4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для забезпечення роботи веб-додатку необхідно наявність мережі Інтернет та перехід на localhost:4200.

При запуску веб –додатку користувач бачить перед собою екран авторизації та реєстрації.

Якщо користувач незареєстрований необхідно обрати вкладку ‘Registration’ та заповнити усі поля, після чого натиснути кнопку ‘Register’ (рисунок 4.1).

Якщо користувач уже зареєстрований в системі необхідно обрати вкладку ‘Authorization’(рисунок 4.2), заповнити поля логін та пароль, після чого натиснути кнопку ‘Log In’.



The image shows a web application interface for 'KPI playground'. At the top is a teal header with the text 'KPI playground'. Below it is a light gray area containing a registration form. The form has two tabs: 'Authorization' and 'Registration', with 'Registration' being the active tab. The form is titled 'Register' and contains the following fields: 'Name' (with the value 'John'), 'Surname' (with the value 'Doe'), 'Email' (with the value 'johndoe@gmail.com'), 'Password' (with masked characters '....'), and 'Confirm Password' (with masked characters '...'). There is a checkbox labeled 'Student' which is currently unchecked. At the bottom of the form is a green button labeled 'Register'. The footer of the application is a dark gray bar with the text 'Playground KPI' and '©All rights reserved. 2019'.

Рисунок 4.1 — Екран реєстрації

The screenshot shows a web interface for 'KPI playground'. At the top is a teal header with the text 'KPI playground'. Below it is a light gray area containing a white form with two tabs: 'Authorization' (selected) and 'Registration'. The form is titled 'Log in'. It has a label 'Login (Your Email)' above a text input field containing 'johndoe@gmail.com'. Below this is a 'Password' label above a text input field with four dots. A green 'Log in' button is at the bottom of the form. At the very bottom of the page is a dark gray footer with the text 'Playground KPI' and '©All rights reserved, 2019'.

Рисунок 4.2 — Екран авторизації

В випадку неправильного заповнення форми, чи незаповнення певних обов'язкових полів, користувачеві виводяться повідомлення про помилку та кнопка авторизації чи реєстрації буде неактивною (рисунок 4.3).

The screenshot shows the 'Registration' tab of the 'KPI playground' form. The form is titled 'Register'. It has several input fields: 'Name' (filled with 'John'), 'Surname' (empty), 'Email' (filled with 'johndoe@gmail.com'), 'Password' (filled with six dots), and 'Confirm Password' (filled with six dots). Below the 'Surname' field is a red error message: '* Surname is obligatory field'. Below the 'Confirm Password' field is a yellow error message: 'Password and Confirm Password must be the same'. There is a checked radio button for 'Student' and an empty 'Student Ticket' field. A gray 'Register' button is at the bottom. The 'Authorization' tab is also visible on the left.

Рисунок 4.3 — Неправильне заповнення форми

Якщо користувач не ввів дані в обов'язкове поле, то з'являється червоне повідомлення(error) про те, що поле має обов'язково бути заповнене. Крім того поля пароль та підтвердження паролю, мають бути однаковими. Про це користувач бачить жовте повідомлення (warning), що пароль та підтвердження паролю мають бути однаковими. Якщо обрати чекбокс студент, то з'являється додаткове поле для введенню студенського квитка, при чому при виборі статусу студент, це поле також стає обов'язковим. Кнопка стає активною після правильного введення даних.

Після успішного входу в додаток користувач потрапляє в особистий кабінет. За замовчуванням відкрита вкладка меню – 'Schedule' (риисунок 4.4).

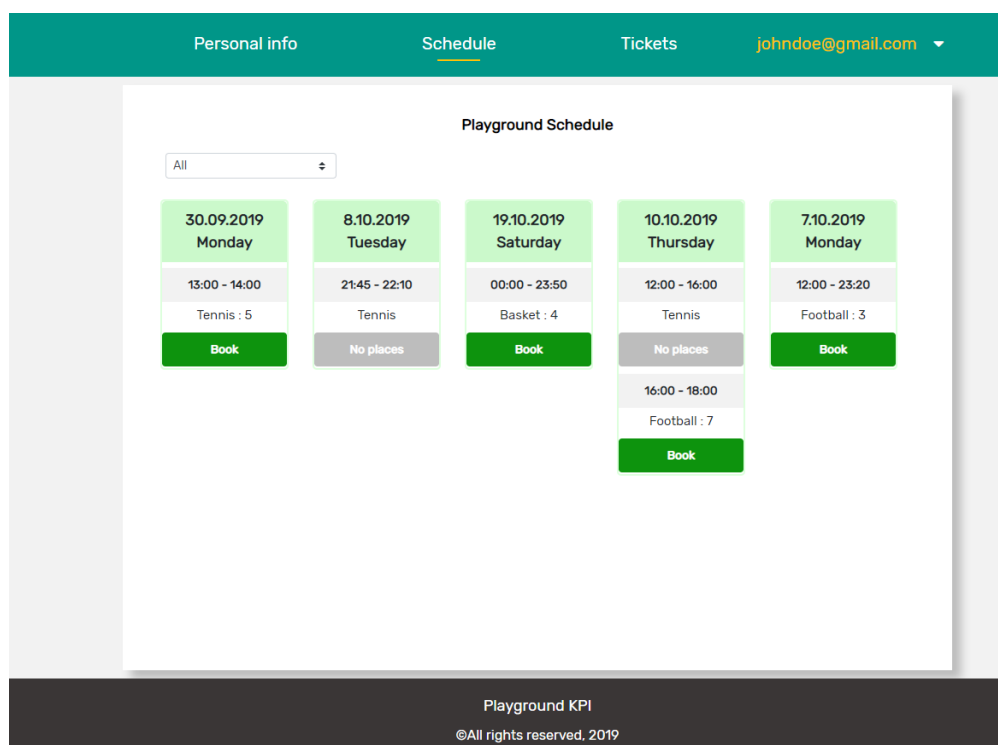


Рисунок 4.4 — Розклад

На вкладці 'Schedule' відображається актуальний розклад занять. Також можна обирати вид спорту та фільтрувати розклад за обраним видом. Інформація в розкладі відображається за днями. В кожного дня є список заходів з інформацією щодо них, а саме деталі щодо наявності вільних місць. Якщо захід має вільні місця то користувач бачить кнопку 'Book', яку він може натискати, якщо немає – 'No places', яка є неактивною. Після натискання на кнопку 'Book', з'являється діалогове

вікно з деталями заходу та запитом на підтвердження бронювання (рисунок 4.5). Після підтвердження захід бронюється за даними користувача, у випадку відміни, користувач потрапляє на екран розкладу.

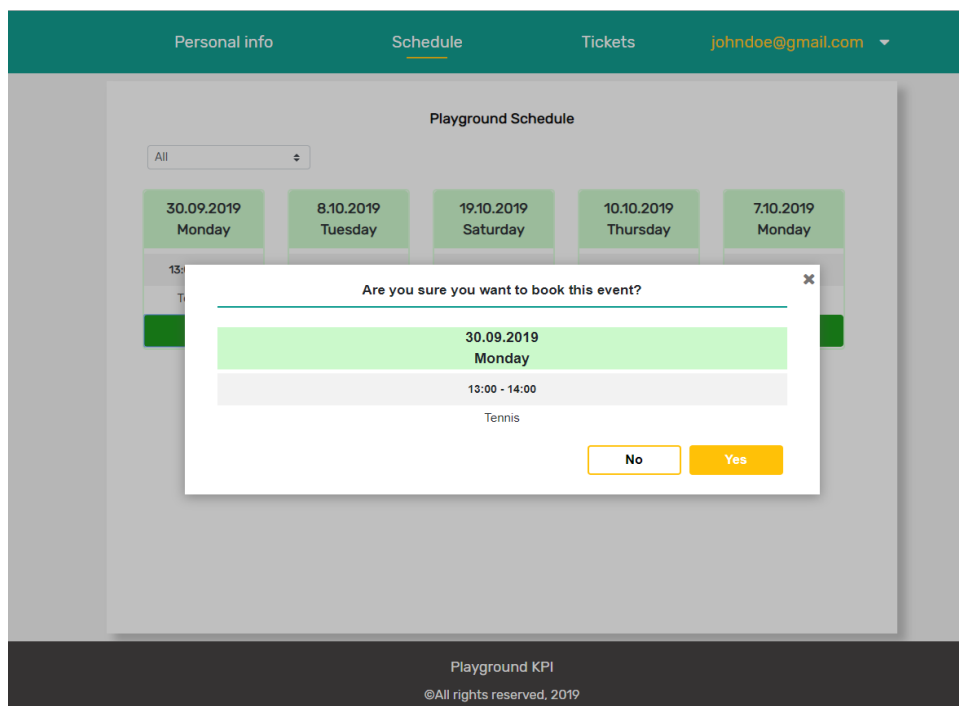


Рисунок 4.5 — Підтвердження бронювання

Після бронювання заходу, актуальні квитки можна знайти на вкладці 'Tickets' (рисунок 6.6).

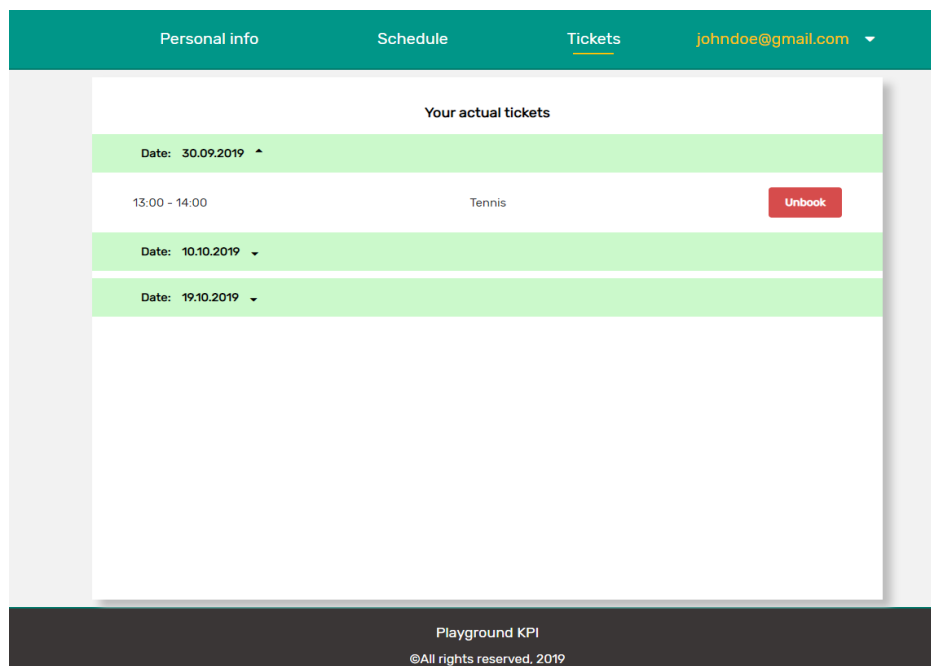


Рисунок 4.6 — Актуальні квитки

Квитки відображаються по днях. Для перегляду квитків на конкретний день, треба обрати день, після чого з'явиться випадаюче вікно з квитками на цей день. В даних квитку знаходиться час та вид спорту, який заброньовано. Для відміни бронювання необхідно натиснути 'Unbook' після чого з'явиться діалогове вікно з підтвердженням відміни бронювання. При підтвердженні бронювання на захід буде скасовано(рисунок 4.7).

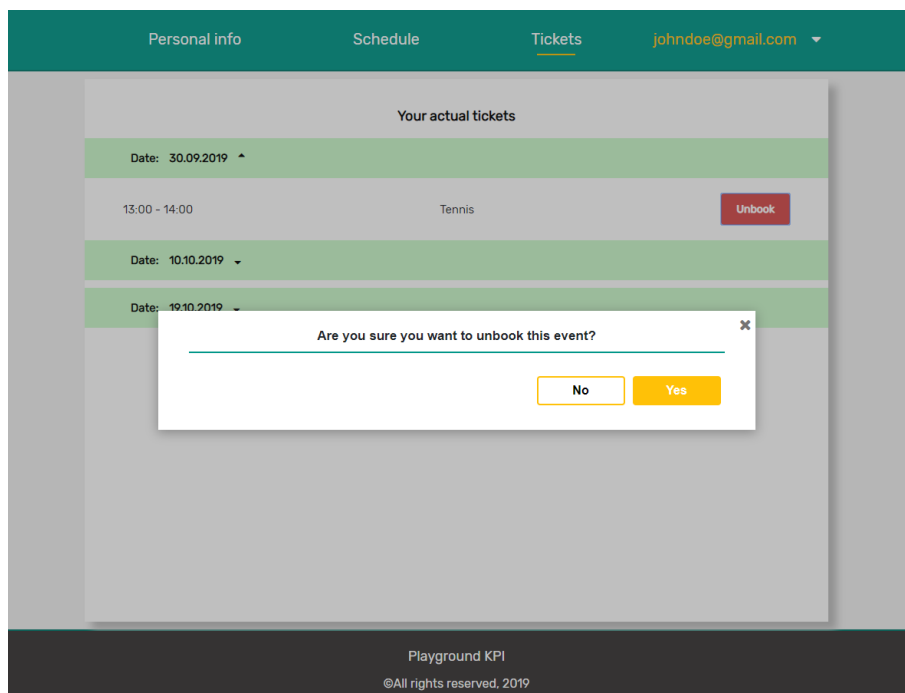


Рисунок 4.7— Скасування бронювання заходу

Особиста інформація користувача знаходиться на вкладці 'Personal Info' (рисунок 4.8).

Інформація може переглядатись або редагуватись. Для редагування інформації необхідно натиснути кнопку 'Edit', після чого ввести необхідні зміни в поля. У випадку, якщо форма заповнена вірно, кнопка 'Save' буде активною, а значить користувач зможе її обрати (рисунок 4.9).

При натисканні на 'Save' дані користувача буде змінено в системі. Якщо натиснути кнопку 'Cancel' зміни не будуть збережені. Якщо при авторизації користувач не вказав студентський квиток, він може його додати при редагуванні особистої інформації.

Personal info Schedule Tickets johndoe@gmail.com ▾

Personal Info Edit

Login (Your Email): johndoe@gmail.com

Name: John

Surname: Doe

Password: 1234

Student's Ticket: KB123123

Playground KPI
©All rights reserved, 2019

Рисунок 4.8 — Особиста інформація користувача

При оновленні сторінки користувача, в випадку якщо користувач не виконував Log out, дані користувача будуть збережені, і при повторному відкритті веб додатку, користувач потрапляє в особистий кабінет, без необхідності повторної авторизації.

Personal info Schedule Tickets johndoe@gmail.com ▾

Editing Personal Info Cancel Save

Login (Your Email): johndoe@gmail.com

Name

Surname

Password

Confirm Password

☒ Student

Student Ticket

Playground KPI
©All rights reserved, 2019

Рисунок 4.9 — Редагування особистої інформації користувача

Для виходу з сторінки користувача необхідно натиснути стрілку біля логіну користувача, що знаходиться в хедер компоненті (рисунок 4.10).

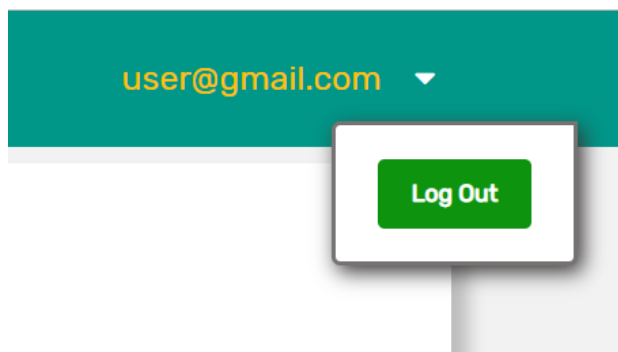


Рисунок 4.10 — Вихід з кабінету користувача

Після чого з’явиться вікно з кнопкою ‘Log out’. Натиснувши на неї користувач виходить з системи, та на екрані з’являється початковий екран з авторизацією та реєстрацією.

5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

З метою сформувати інноваційне та підприємницьке мислення щодо створення, розвитку та поширення власного проекту було вирішено проаналізувати систему “Відкритого спортивного майданчику з е-сервісами” як стартап-проект з можливістю його комерціалізації.

5.1. Опис ідеї стартап-проекту

У даному розділі описано маркетинговий аналіз стартап-проекту “Відкритий спортмайданчик з е-сервісами” для визначення можливості його ринкового впровадження та способів реалізації даного проекту. Метою розділу є формування підприємницького та інноваційного мислення, формування здатностей щодо оцінювання ринкових перспектив і можливостей комерціалізації основних науково-технічних розробок, сформованих у попередній частині магістерської дисертації у вигляді розроблення концепції стартап-проекту в умовах висококонкурентної ринкової економіки глобалізаційних процесів. Опис ідеї стартап-проекту наведено в таблиця 5.1.

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Ідея полягає в тому, щоб створити систему відкритого спортивного майданчику, для контролю та управління доступом відвідувачів	1. Використання додатку для майданчиків на території студмістечка	1. Можливість інтеграції додатку з веб-сервером, який у свою чергу інтегрується з СКУД
	2. Використання додатку для приватних спортивних закладів	2. Можливість збереження особистих даних в особистому

		кабінеті
		3. Можливість перегляду актуальних занять майданчику та кількості вільних місць
		4. Можливість бронювання чи скасування бронювання заходу
		5. Швидкість та кросплатформеність

Таблиця 5.1. Опис ідеї стартап-проекту

Отже, проект Веб-додаток інтернет-сервісу “Відкритий спортмайданчик з е-сервісами” може бути використаним як інструмент для користувача і прошарком в загальній системі, де додаток інтегрується з веб-сервером за принципами REST API.

Для реалізації стартап-проекту необхідно визначити його характеристики (таблиця 5.2).

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слаба сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Конкурент 1	Конкурент 2	Конкурент 3			
1.	Форма виконання	Веб-додаток	Сайт	Сайт	Сайт			+
2.	Собівартість	Низька	Середня	Середня	Середня			+
3.	Наявність адміністратора для	Так	Ні	Ні	Ні		+	

	налаштування							
4.	Збереження особистої інформації	Так	Ні	Ні	Ні			+
5.	Бронювання та скасування бронювання заходів	Так	Ні	Ні	Ні			+
6.	Перегляд актуального розкладу	Так	Ні	Так	Так			+
7.	Кросплатформеність	Так	Так	Так	Так			+
8.	Інтеграція з веб-сервером	Так	Ні	Ні	Так			+
9.	Орієнтована на студентські квитки	Так	Ні	Ні	Ні	+		

Таблиця 5.2 – Визначення сильних, слабких, нейтральних характеристик ідеї проекту

Отже, нейтральними сторонами проекту є те, що системі необхідний адміністратор для налаштування. Слабкою стороною є те, що на даний момент система орієнтована на прохід через студентські квитки, але й майбутньому може бути адаптована до іншого варіанту проходу. Сильними сторонами є отримання актуальних даних, можливість виконувати операції онлайн, кросплатформеність.

5.2. Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення

товару)(таблиця 5.3).

<i>№ п/п</i>	<i>Ідея проекту</i>	<i>Технології її реалізації</i>	<i>Наявні сть техно логій</i>	<i>Доступ ність техноло гій</i>
1.	Створення веб- додатку	Angular Framework, TypeScript, JavaScript	Наявна	Безкоштовна, доступна
		NPM packages	Наявна	Частково безкоштовна, доступна
		RxJS	Наявна	Безкоштовна, доступна
		HTML, SCSS	Наявна	Безкоштовна, доступна
3.	Збереження додатку в віддаленому репозиторії	Git	Наявна	Безкоштовна, доступна
		Github	Наявна	Безкоштовна, доступна

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

Обрані технології реалізації ідеї проекту: Angular Framework, TypeScript, JavaScript через повну безкоштовність та доступність, наявність докладної документації; NPM packages через часткову безкоштовність, але було використано та завантажено пакети, яке є в безкоштовними. Git, Github через безкоштовність та можливість збереження програмного коду додатку на віддаленому репозиторії, та контролю над операціями з кодом.

5.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового

впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів(таблиця 5.4).

<i>№ п/п</i>	<i>Показники стану ринку (найменування)</i>	<i>Характеристика</i>
1.	Кількість головних гравців, од	5
2.	Загальний обсяг продаж, грн/ум.од	8000 грн./ум.од
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Немає
5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	$R = (3000000 * 100) / (1000000 * 12) = 25\%$

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

Отже, середня норма рентабельності в галузі менша, ніж банківський відсоток на вкладення. Тому має сенс вкласти кошти в саме цей проект, адже проект немає наявності обмеження для входу на ринок і специфічних вимог до стандартизації та сертифікації, бо він буде виконаний у вигляді веб-додатку додатку.

Далі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (таблиця 5.5).

<i>№ п/п</i>	<i>Потреба, що формує ринок</i>	<i>Цільова аудиторія (цільові сегменти ринку)</i>	<i>Відмінності у поведінці різних потенційних цільових груп клієнтів</i>	<i>Вимоги споживачів до товару</i>
1.	Веб-додаток, що	Потенційними	Цільова група	Рішення повинне

	надає доступ до інформації щодо майданчику	цільовими групами є університети та інші навчальні заклади, які мають спортивний майданчик на території закладу, а також інші приватні спортивні заклади	визначає правила доступу на майданчик та необхідність оплати	бути зручним у користуванні, кросплатформеним, швидкодійним
--	--	--	--	---

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

Визначено характеристики стартап-проекту: основну потребу, що формує ринок – вед-додаток; наведено основні цільові сегменти ринку — університети та інші навчальні заклади, які мають спортивний майданчик на території закладу, а також інші приватні спортивні заклади; відмінності у поведінці різних потенційних цільових груп клієнтів — в визначенні правил доступу на майданчик та необхідності оплати; затверджено основні вимоги до споживачів – вед-додаток повинен бути зручним у користуванні, кросплатформеним та швидкодійним.

Далі складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (таблиця 5.6-5.7).

<i>№ п/ п</i>	<i>Фактор</i>	<i>Зміст загрози</i>	<i>Можлива реакція компанії</i>
1.	Конкуренція	Вихід на ринок великої компанії	1. Вихід з ринку 2. Передбачити додаткові переваги власного ПЗ для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок
2.	Зміна потреб користувачів	Користувачам необхідний додаток з іншим функціоналом	Передбачити можливість додавання нового функціоналу до створеного ПЗ

3.	Надходження на ринок альтернативних продуктів	Перехід користувачів нашого товару на інший продукт	Впровадження нового функціоналу, якого немає у конкурентів
4.	Зменшення кількості користувачів веб додатку	Зменшення зацікавленості користувачів в додатку	Вчасно оновлювати інтерфейс додатку
5.	Уповільнення росту ринку	Скорочення користувачів продуктів, що тільки виходять на ринок	Інвестиції у впровадження ефективної реклами продукту

Таблиця 5.6 — Фактори загроз

Було наведено основні фактори загроз стартап-проекту.

Найбільшою загрозою для проекту є конкуренція (вихід на ринок великої компанії) і зміна потреб користувачів (користувачам необхідний додаток з іншим функціоналом).

Для зменшення цих загроз потрібно передбачити додаткові переваги власного додатку для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок, а також передбачити можливість додавання нового функціоналу до створюваного додатку і своєчасного оновлення програмного забезпечення і користувацького інтерфейсу, а також передбачити можливість інвестицій в рекламу продукту в випадку уповільнення росту ринку.

<i>№ п/ п</i>	<i>Фактор</i>	<i>Зміст можливості</i>	<i>Можлива реакція компанії</i>
1.	Зростання можливостей потенційних покупців	Зростання держфінансування досліджень у галузі спортивних закладів	Змога запропонувати продукт більшої кількості потенційних користувачів

2.	Зниження довіри до конкурента	Неактуальність інформації конкурентів	При виході на ринок звертати увагу покупців на достовірність нашого додатку та актуальність інформації
3.	Стрімке зростання росту ринку	Компаніям, що тільки виходять на ринок, буде простіше отримати клієнтів	Змога запропонувати продукт більшої кількості потенційних користувачів
4.	Обслуговування додаткових груп споживачів	Поява нових потенційних груп споживачів	Змога розширити продукт для подальшого впровадження у нові галузі
5.	Розширення асортименту послуг	Поява нового функціоналу, що привабить нових користувачів	Розробка нового функціоналу, що є потребою певної групи користувачів

Таблиця 5.7 – Фактори можливостей

Було наведено основні фактори сприяння ринковому впровадженню проекту: зростання можливостей потенційних покупців, зниження довіри до конкурента, стрімке зростання росту ринку, обслуговування додаткових груп користувачів, розширення асортименту послуг. Основними реакціями компанії є: змога запропонувати продукт більшої кількості потенційних користувачів, звернути увагу покупців на достовірність нашого додатку та актуальність інформації, розробка нового функціоналу, що є потребою певної групи користувачів та розширення інтерфейсу та функціоналу продукту для подальшого впровадження у нові галузі.

Надалі визначаються загальні риси конкуренції на ринку (таблиця 5.8).

<i>Особливості конкурентного середовища</i>	<i>В чому проявляється дана характеристика</i>	<i>Вплив на діяльність підприємства (можливі дії компанії, щоб бути</i>
---	--	---

		конкурентоспроможн ою)
1. Вказати тип конкуренції - досконала	Існує 3 фірми-конкурентки на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2. За рівнем конкурентної боротьби - міжнародний	Одна з компаній – з іншої країни, дві – з України	Додати можливість вибору мови додаток, щоб легше було у майбутньому вийти на міжнародний ринок
3. За галузевою ознакою - внутрішньогалузева	Конкуренти мають додаток, яке використовується лише всередині даної галузі	Створити основу додаток таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях
4. Конкуренція за видами товарів: - товарно-видова	Види товарів є однаковими, а саме – програмне забезпечення	Створити ПЗ, враховуючи недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж використовують конкуренти
6. За інтенсивністю — не марочна	Бренди відсутні	-

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

У Таблиці 5.8 наведено ступеневий аналіз конкуренції на ринку, де було визначено особливості конкурентного середовища та їх вплив на діяльність підприємства. Для досягнення конкурентоспроможності основним фактором є

розробка основи ПЗ таким чином, щоб можна було легко переробити дане ПЗ та розширити для використання у інших галузях.

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі за М.Портером(таблиця 5.9).

<i>Складові аналізу</i>	<i>Прямі конкуренти в галузі</i>	<i>Потенційні конкуренти</i>	<i>Постачальники</i>	<i>Клієнти</i>	<i>Товари-замінники</i>
	<i>Навести перелік прямих конкурентів</i>	<i>Визначити бар'єри входження в ринок</i>	<i>Визначити фактори сили постачальників</i>	<i>Визначити фактори сили споживачів</i>	<i>Фактори загроз з боку замінників</i>
Висновки:	Існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 3, так як його рішення також представлене у вигляді додатку, який інтегрований з сервером	Так, можливість для входу на ринок є, бо наше рішення спрощує та пришвидшує роботу користувача	Постачальник и відсутні.	Важливим для користувача є швидкість роботи ПЗ	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Отже, з огляду на конкурентну ситуацію аналізів за М. Портером можна з впевненістю сказати, що проект має можливість роботи на ринку, тому що серед наведених конкурентів немає тих, які б могли його потіснити, адже розроблене рішення спрощує та пришвидшує роботу користувача.

Можна виділити основні сильні сторони продукту, які б допомогли стати конкурентоспроможним на ринку — кросплатформеність, зручність користувацького інтерфейсу, простота у використанні.

На основі аналізу висновків, наведених вище, визначається та обґрунтовується перелік факторів конкурентоспроможності (таблиця 5.10).

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Можливість виконання операцій онлайн розкладу онлайн	Дозволяє користувачам онлайн бронювати чи скасувати бронювання квитків
2.	Збереження даних в особистому кабінеті користувача	Дозволяє користувачам авторизуватись та бронювати заняття, без повторного введення даних
3.	Кросплатформеність та швидкість	Доступ відбувається через веб-додаток користувача
4.	Актуальність інформації	Отримання даних з серверу

Таблиця 5.10 — Обґрунтування факторів конкурентоспроможності

Було наведено основні фактори конкурентоспроможності, які будуть представлені на ринку, а саме: надання можливості виконання операцій онлайн, кросплатформеність та швидкість веб-додатку та отримання оновленої та актуальної інформації.

За визначеними факторами конкурентоспроможності проводиться аналіз сильних та слабких сторін стартап-проекту (таблиця 5.11).

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з нашим підприємством						
			-3	-2	-1	0	1	2	3
1.	Можливість операцій онлайн	20	+						
2.	Отримання актуальної інформації	20		+					

3.	Кроссплатформеність	15				+			
4.	Швидкість	15				+			

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін проекту

Було наведено порівняльний аналіз сильних сторін проекту товарів-конкурентів і нашого підприємства.

Найбільше балів набрано для таких факторів конкурентноспроможностей — можливість виконання операцій онлайн, отримання актуальної інформації, найменше – у кроссплатформеності та швидкості.

Далі складаємо SWOT-аналіз (таблиця 5.12).

Сильні сторони: швидкість , кроссплатформеність, можливість виконання операцій онлайн, отримання актуальної інформації	Слабкі сторони: пропуск за студенським квитком
Можливості: стрімкий ріст попиту на веб-додаток через оновлення майданчиків, можливість стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг	Загрози: конкуренція, зміна потреб користувачів, надходження на ринок альтернативних продуктів, уповільнення росту ринку, зменшення кількості користувачів веб-додатку

Таблиця 5.12. SWOT- аналіз стартап-проекту

Отже, внутрішні можливості компанії і спроможності щодо виведення продукту на ринок характеризуються такими сильними і слабкими сторонами: сильні—кроссплатформеність та швидкість, можливість виконання операцій онлайн; слабкі — наразі пропуск на майданчик відбувається через студентський квиток. Ринкові можливості компанії щодо зовнішнього оточення характеризуються можливостями і загрозами: можливості – стрімкий ріст попиту на веб-додаток через оновлення майданчиків, можливість стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг, загрози — конкуренція, зміна потреб користувачів, надходження на ринок альтернативних

продуктів, уповільнення росту ринку, зменшення кількості користувачів веб-додатку.

На основі SWOT-аналізу складаються альтернативи ринкового впровадження стартап-проекту (таблиця 5.13).

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення додатку з використанням фреймворке Angular	90%	3 місяці
2.	Створення програми на основі без використання будь-яких фреймворків для обробки даних	35%	8 місяців

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

Отже, було визначено альтернативу ринкового впровадження стартап-проекту — створення веб-додатку на основі технологій, адже для неї отримання ресурсів є більш простими та гнучким, а строки реалізації — меншими.

5.4. Розробка ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (таблиця 5.14).

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Просто та входу у сегмент
-------	--	---	---	--------------------------------------	---------------------------

1.	Навчальні заклади	Контроль доступу студентів на майданчик	Великий	Існує 3 конкуренти, які надають схожі, але більш вузькі і дорогі рішення.	Можливість інтеграції в існуючу систему, швидкодія, Кросплатформеність Онлайн виконання операцій
2.	Приватні спортивні заклади	Контроль доступу відвідувачів	Великий		Можливість інтеграції в існуючу систему, швидкодія, Кросплатформеність Онлайн виконання операцій
Які цільові групи обрано: навчальні заклади та приватні спортивні заклади					

Таблиця 5.14 — Вибір цільових груп потенційних споживачів

Отже, було вибрано основні цільові групи: навчальні заклади та приватні спортивні заклади адже їм необхідно контролювати потік відвідувачів майданчику.

Також було визначено основні характеристики для входу в сегмент: швидкодія, кросплатформеність, можливість інтегрування в існуючу систему, виконання операцій онлайн.

Отже, проілюструвати базову стратегію розвитку можна у вигляді таблиці 5.15

<i>№ n/ n</i>	<i>Обрана альтернатива розвитку проекту</i>	<i>Стратегія охоплення ринку</i>	<i>Ключові конкурентоспро можні позиції відповідно до обраної альтернативи</i>	<i>Базова стратегія розвитку</i>
1.	Створення веб-додатку, використовуючи Angular, TypeScript	Ринкове позиціонування	Можливість інтеграції в уже існуючі системи, кросплатформеність, швидкодія, онлайн виконання операцій	Диференціація

Таблиця 5.15 – Визначення базової стратегії розвитку

Було обрано таку альтернативу розвитку проекту: створення веб-додатку, використовуючи Angular, TypeScript, адже завдяки цим технологіям можна досягнути ключових конкурентноспроможних позицій кінцевого продукту.

Наступним кроком є вибір стратегії конкурентної поведінки (таблиця 5.16).

<i>№ n/n</i>	<i>Чи є проект «першопрохідцем» на ринку?</i>	<i>Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?</i>	<i>Чи буде компанія копіювати основні характеристики товару конкурента, і які?</i>	<i>Стра тегія конку рент ної повед інки</i>

1.	Ні	Так	Буде, а саме: основною задачею є розробка веб-додатку з базовою інформацією майданчику(конкуренти 1, 2, 3), інтегрованим з сервером(конкурент 3) та доданим функціоналом авторизації, реєстрації та бронювань	Зайняття конкурентної ніші
----	----	-----	--	----------------------------

Таблиця 5.16 — Визначення базової стратегії конкурентної поведінки

Отже, було визначено базову стратегію конкурентної поведінки — зайняття конкурентної ніші, адже програмний продукт буде націлений на один ринковий сегмент. Також матиме можливість авторизації, реєстрації, бронювань — основну перевагу перед конкурентами, що формуватиме довіру і прихильність споживачів.

Далі визначається стратегія позиціонування проекту, яка допоможе користувачам ідентифікувати програмний продукт (табл. 5.17).

<i>№ п/п</i>	<i>Вимоги до товару цільової аудиторії</i>	<i>Базова стратегія розвитку</i>	<i>Ключові конкурентоспроможні і позиції власного стартап-проекту</i>	<i>Вибір асоціацій, які мають сформулювати комплексну позицію власного проекту</i>
1.	Простота інтерфейсу, швидкодія, можливість виконання операцій онлайн, надійність	Диференціація	Можливість інтеграції в уже існуючі системи, виконання операцій онлайн та отримання актуальних даних дозволить відслідковувати події в режимі реального часу	Швидкодія, простота платформенної архітектури, простота та актуальність даних

Таблиця 5.17 — Визначення стратегії позиціонування

Отже, було визначено стратегію позиціонування, а саме визначено основні вимоги до товару цільової аудиторії: швидкодія, виконання операцій онлайн, кроссплатформеність; базову стратегію розвитку: диференціація; ключові конкурентоспроможні позиції стартап-проекту: можливість інтеграції в уже існуючі системи, отримання актуальних даних дозволить отримувати необхідні дані і відслідковувати події в режимі реального часу, виконання операцій оналайн.

5.5. Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у таблиці 5.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

<i>№ п/п</i>	<i>Потреба</i>	<i>Вигода, яку пропонує товар</i>	<i>Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)</i>
1.	Отримання актуальних даних	Додаток інтегрований з веб-сервером за RESTful принципами	Перевага в універсальності на можливості інтегрувати додаток у існуючі системи.
2.	Збереження особистих даних	Збереження даних користувача, забезпечення авторизації та реєстрації. Збереження входу на сторінку користувача при перезавантаженні вікна чи випадковому закритті вкладки браузера	Швидкий доступ в систему, відсутність необхідності постійно вводити особисті дані для бронювання події
3.	Перегляд розкладу	Перегляд актуального розкладу у вигляді таблиць	Розклад у вигляді таблиць з

			фільтрами
4.	Виконання операцій онлайн	Можливість бронювати чи скасовувати бронювання через веб-інтерфейс	Веб-інтерфейс для бронювання

Таблиця 5.18 — Визначення ключових переваг концепції потенційного товару

Отже бачимо, що проект має ключові переваги перед конкурентами, які повністю відповідають потребам цільової аудиторії. Додаток реалізований у вигляді веб-додатку, що є універсальним для подальшої інтеграції сервісу в інші системи та надає веб-інтерфейс для перегляду актуальних даних майданчику щодо розкладу. Надалі розробляється трирівнева маркетингова модель товару (таблиця 5.19).

Рівні товару	Сутність та складові		
I. Товар за задумом	Веб-додаток, що інтегрований з веб-сервером та надає можливість виконання операцій онлайн		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1.Онлайн виконання операцій 2. Отримання актуальної інформації 3. Швидкість роботи 4. Кросплатформеність	1.Нм 2.Нм 3.Нм 4.Нм	1.Технологічна 2.Технологічна 3.Технологічна 4.Технологічна
	Якість: згідно до стандарту ISO 4444 буде проведено тестування		
	Маркування відсутнє		
	Моя компанія: “Engineering future”		
III. Товар із підкріпленням	1-місячна пробна безкоштовна версія		
	Постійна підтримка для користувачів		
За рахунок чого потенційний товар буде захищено від копіювання: патент			

Таблиця 5.19 — Опис трьох рівнів моделі товару

Було описано три рівні моделі товару, з чого можна зробити висновок, що основні властивості товару у реальному виконанні є нематеріальними та технологічними. Також було надано сутність та складові товару у задумці та товару з підкріпленням. Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. У даному випадку найбільш вірогідним гарантом буде патент.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (таблиця 5.20). Аналіз проводиться експертним методом.

<i>№ п/п</i>	<i>Рівень цін на товари-замінники, грн.</i>	<i>Рівень цін на товари-аналоги, грн.</i>	<i>Рівень доходів цільової групи споживачів, грн.</i>	<i>Верхня та нижня межі встановлення ціни на товар/послугу, грн.</i>
1.	40000	35000	120000	35000-40000

Таблиця 5.20 — Визначення меж встановлення ціни

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (таблиця 5.21).

<i>№ п/п</i>	<i>Специфіка закупівельної поведінки цільових клієнтів</i>	<i>Функції збуту, які має виконувати постачальник товару</i>	<i>Глибина каналу збуту</i>	<i>Оптимальна система збуту</i>
1.	Придбання підписки та оплата щомісячних внесківпродовження ліцензії	Продаж	0(напрямую), 1(через одного посередника)	Власна та через посередників

Таблиця 5.21 — Формування системи збуту

Тому, система приносить прибуток завдяки щомісячним внескам для продовження ліцензії та придбанням підписок.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 5.22).

<i>№ п/ п</i>	<i>Специфіка поведінки цілових клієнтів</i>	<i>Канали комунікацій, якими користують ся цілові клієнти</i>	<i>Ключові позиції, обрані для позиціон ування</i>	<i>Завдання рекламног о повідомле ння</i>	<i>Конце пція реклам ного зверне ння</i>
1.	Придбання ліцензії на користування в мережі Інтернет, щомісячне її продовження, користування на власних серверах.	Інтернет	Авторизація, реєстрація, бронювання, надійність, швидкість роботи	Показати переваги сервісу, у тому числі і перед конкурентами	Демо-ролик із Використанням

Таблиця 5.22 — Концепція маркетингових комунікацій

Отже, в Таблиці 5.22 наведено концепцію маркетингових комунікацій, було визначено, що придбання ліцензії на користування буде здійснюватись в мережі Інтернет, необхідним буде щомісячне її продовження, користування додатком на власних серверах. Концепцією рекламного звернення буде демо-ролик з використанням додатку, у якому буде показано переваги сервісу перед конкурентами.

5.6. Результати аналізу стартап ідеї

У даному розділі були досліджені основні аспекти виходу на ринок веб-додатку для “Відкритого спортивного майданчику з е-сервісами”. Описаний продукт є доцільним для користувачів, які хочуть мати доступ до актуальної інформації спортивного майданчику, а також бути активним його учасником.

В даному розділі було проаналізовано сильні та слабкі характеристики проекту для формування його конкурентоспроможності; також обрано технологію реалізації ідеї проекту: для створення веб-додатку обрано фреймворк Angular, який є безкоштовним та з відкритою документацією; було проведений аналіз конкуренції на ринку, SWOT аналіз та розглянуті альтернативи впровадження стартап проекту; обґрунтовані фактори конкурентоспроможності; проведений менеджмент потенційних ризиків; в додаток до цього було сформовано маркетингову програму для зацікавлення майбутніх користувачів в даному продукті.

Отже, відповідно до проведених досліджень існує можливість ринкової комерціалізації проекту та перспективи впровадження з огляду на потенційні групи клієнтів, бар'єри входження не є високими, проект має значні такі особливості як кросплатформеність та швидкодія, а також переваги перед конкурентами в наявності актуальної інформації та можливості виконання операцій онлайн. Для успішного виконання проекту необхідно реалізувати додаток із використанням фреймворку Angular. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація та впровадження проекту є доцільними.

ВИСНОВКИ

Отже, проблема надання автоматизованих сервісів в сферу контролю та управління доступу спортивних комплексів є актуальною в даний час. Для можливості контролю навантаження закладів та збереження відвідувачів, було створено систему “Відкритого спортивного майданчику з е-сервісами”.

При побудові архітектури для системи мною було обрано сервіс-орієнтований підхід, що означає, що кожна її складова відповідає за окремі бізнес-функції та їх розробка не впливає на зміни в розробці іншої підсистеми.

На даний момент розроблювалась базова частина – система контролю та управління доступу на спортивний майданчик, яка складається з веб-застосунку, серверу та СКУД. Надалі система відкритого спортивного майданчику може бути доповнена системами суддівства, системою видачі інвентаря, системою освітлення та інші.

Інструментом, який надає керування доступом на майданчик є клієнтська частина, яка взаємодіє з серверним застосунком.

В ході роботи було проаналізовано існуючі застосунки спортивних закладів, проаналізовано їх переваги на обмеження та визначено вимоги до створюваної системи.

Основними задачами системи є забезпечення кросплатформеності та швидкодії, забезпечення користувача актуальною інформацією та надання можливості виконання онлайн основних операцій спортивного майданчику, таких як авторизація, реєстрація, бронювання, скасування бронювання події. Відповідно до поставлених задач, було проаналізовано сучасні технології створення застосунків та обрано такі, які допоможуть вирішити поставлені в ході роботи задачі.

Для вирішення задачі кросплатформеності було вирішено обрати веб-застосунок, адже таким чином користувачі можуть отримувати доступ до системи з будь-якого приладу за наявності мережі Інтернет.

Для вирішення задачі швидкодії обрано фреймворк Angular, який дозволяє створювати великі динамічні веб-застосунки, які швидко завантажуються та реагують на дії користувача, крім того було додано бібліотеку RxJS з метою додавання асинхронного програмування, що також прискорює та не блокує роботу клієнта з веб-застосунком. Також при розробці було вирішено використати мову TypeScript, яка розширює мову JavaScript.

Для вирішення задачі забезпечення користувача актуальною інформацією та виконання операцій майданчику онлайн, було вирішено створити комунікацію веб-додатку з серверним додатком на основі власного прикладного користувацького інтерфейсу.

Архітектура веб-додатку була визначена відповідно до основних складових фреймворку Angular, а саме модулі, компоненти та сервіси.

Також, проаналізовано можливість комерціалізації проекту, що показує, що проект може бути успішно впроваджений в сучасних умовах ринку.

Обмеженнями застосунку є те, що наразі система була орієнтована на студентів, тому прив'язана до студентських квитків, а також те, що додаток не розгорнутий на сервері, а запускається локально. Це є першочергові кроки до поліпшення системи.

Описана методика розробки веб-додатку дозволяє писати великі веб-застосунки, які можуть бути з легкістю розширені в майбутньому. В випадку необхідності розширення застосунку необхідно створити нову гілку з використанням системи контролю версій, додати нові модулі в Angular проекті з необхідними компонентами та сервісами. Після чого, після успішного завершення розробки та тестування, нова функція може бути додана в основну робочу версію проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Советов Б. Архитектура информационных систем / Б. Советов, А. Водяхо., 2012.
2. Bell M. Introduction to Service-Oriented Modeling / Michael Bell., 2008.
3. Весперман Д. Управление версиями та управління вихідним кодом / Дженіфер Весперман. – Tokyo: O`Reilly Media Inc.. – 430 с.
4. Hutten D. Git: Learn Version Control with Git: A step-by-step Ultimate beginners Guide / Dennis Hutten., 2019.
5. Loeliger J. Version Control with Git: Powerful tools and techniques for collaborative software development 2nd Edition, Kindle Edition / J. Loeliger, M. McCullough, 2017.
6. Чакон С. Git для професійного програміста / С. Чакон, Б. Штрауб., 2016.
7. Mead A. Learning Node.js Development / Andrew Mead., 2018.
8. Syed B. Beginning Node.js / Basarat Syed., 2014. – 310 с.
9. Маккоу А. Веб-додатки на JavaScript. Москва: издательство “Питер”, 2012.
10. Крокфорд Д. JavaScript: сильні сторони/ Д. Крокфорд., 2012. – (Питер).
11. Huber T. C. Getting Started with TypeScript: Includes Introduction to Angular / Thomas Claudius Huber., 2017.
12. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale / Boris Cherny. – United States of America, 2019. – 324 с.
13. Міковський М. Розробка односторінкових веб-додатків / Майкл Міковський., 2014.
14. Еміт С. SPA дизайн та архітектура: розуміння веб-додатків на одній сторінці / Скот Еміт. – Shelter Island: Manning Publications Co., 2016.
15. Козловский П., Питер Бекон Дарвін. Разработка вебприложений с использованием AngularJS. – ДМК Пресс, 2014.
16. Monteiro F. Learning Single-page Web Application Developmen // Monteiro F. — Packt Publishing, 2014 – 214 с.

17. Vepsäläinen J. SurviveJS — Webpack / Juho Vepsäläinen., 2017.
18. Ніксон Р. Створюємо динамічні сайти за допомогою PHP, MySQL, JavaScript, CSS и HTML5, 2016.
19. Майер Е. А. CSS-каскадні стилі таблиць / Е. А. Майер., 2006.
20. Frain B. Sass and Compass for Designers / Ben Frain.. – 276 с.
21. Bradley S. Sass for Beginners: How to Write More Organized and Maintainable Stylesheets / Steven Bradley. – Colorado: Vanseo Design, 2018.
22. Noring C. Architecting Angular Applications with Redux, RxJS, and NgRx: Learn to build Redux style high-performing applications with Angular 6 / Christoffer Noring. – Mumbai: packt, 2018.
23. Farhi O. Reactive Programming with Angular and ngrx: Learn to Harness the Power of Reactive Programming with RxJS and ngrx Extensions / Oren Farhi. – Israel: apress, 2017. – 168 с.
24. Фрімен Е. Патерни проектування / Ерік Фрімен. – Пітер, 2014. – 656 с.
25. Лаурет А. Дизайн Web APIs / Arnaud Lauret. – США: Manning Publications Co., 2019.
26. Річардсон Л. RESTful Web APIs: сервіси для зміни світу / Л. Річардсон, М. Амундсен. – США: O'Reilly Media Inc., 2013.
27. Masse M. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces / Mark Masse. – Tokyo: O'Reilly Media Inc.. – 116 с.
28. Freeman A. Essential TypeScript: From Beginner to Pro / Adam Freeman. – London: apress. – 546 с.
29. Freeman A. Pro Angular 6 / Adam Freeman. – London: apress, 2018. – 804 с.
30. Туловский А. Інструкція розробника Angular, 2013.
31. Zama K. M. Angular Projects: Build nine real-world applications from scratch using Angular 8 and TypeScript / Khan Mohammed Zama. – Mumbai: packt, 2019.
32. Lim G. Beginning Angular with Typescript (updated to Angular 6) / Greg Lim., 2017.

ДОДАТОК А

Веб-додаток інтернет-сервісу “Відкритий спортмайданчик з е-сервісами”

Апробація

УКР.НТУУ “КПІ” ім. І. Сікорського. ТІ41141

Аркушів 7

2019

Факультет інформаційних технологій та управління
Кафедра комп'ютерних наук і математики

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ – 2019

**Збірник тез
VI Всеукраїнської науково-практичної конференції молодих
науковців**

16 травня 2019 року
м. Київ

Київ – 2019

ЗМІСТ

Секція 1. ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ В ОСВІТІ ТА НАУЦІ.....	3
FEATURES OF INTRODUCTION OF INFORMATION TECHNOLOGIES INTO THE EDUCATIONAL PROCESS OF INSTITUTIONS OF HIGHER EDUCATION	
Vakulenko D., Dorenskyi O.	3
ВИКОРИСТАННЯ PLANNER В ОСВІТНЬОМУ ПРОЦЕСІ	
Авраменко І., Євченко Д., Онофрійчук О., Сіренко А.	5
ВЕБ-ДОДАТОК ІНТЕРНЕТ СЕРВІСУ “ВІДКРИТИЙ СПОРТ-МАЙДАНЧИК З Е-СЕРВІСАМИ”	
Амброс С. М., Ковальчук А. М.	6
ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У ЯКОСТІ СУЧАСНОЇ АЛЬТЕРНАТИВИ ЕКСПЕРИМЕНТАМ НАД ТВАРИНАМИ	
Андрійчук М.Д., Сапсай Т.В.	7
РОЗВИТОК ЛОГІЧНОГО МИСЛЕННЯ ШКОЛЯРІВ ЗА ДОПОМОГОЮ ІТ ТЕХНОЛОГІЙ	
Базилик А.В.	9
ПОНЯТТЯ «СТАДІЇ» ТА «ЕТАПИ» У ПРОЕКТУВАННІ ЕЛЕКТРОННИХ ОСВІТНІХ РЕСУРСІВ	
Балалаєва О. Ю.	11
LEGO-EDUCATIONAL ЯК СКЛАДОВА STEM-ОСВІТИ	
Баркалова Т.	13
ВИКОРИСТАННЯ ЗАСОБІВ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ФОРМУВАННЯ ПРОФЕСІЙНИХ КОМПЕТЕНТНОСТЕЙ	
Бірюкова Т.В., Олар О.І.	15
ВИКОРИСТАННЯ СЕРВІСУ QUIZALIZE У ПРОФЕСІЙНІЙ ДІЯЛЬНОСТІ	
Бодненко Д.М., Борисюк А. А., Дерев’яженко Д. І., Калешук М. А., Мозгова А. В., Селецький П. А.	16
ВИКОРИСТАННЯ СЕРВІСУ CALAMEO У ПРОФЕСІЙНІЙ ДІЯЛЬНОСТІ	
Бодненко Д.М., Гавриловська О., Дерменжи Н., Павленко Н.	17

ВЕБ-ДОДАТОК ІНТЕРНЕТ СЕРВІСУ «ВІДКРИТИЙ СПОРТ-МАЙДАНЧИК З Е-СЕРВІСАМИ»

Амброс С. М., Ковальчук А. М.

*Національний технічний університет України "Київський політехнічний інститут
імені Ігоря Сікорського", м. Київ*

Студенти полюбляють спорт, але часом через навантажений навчанням або певними проблемами графік, вони можуть не встигати відвідувати спортивні секції, через незнання розкладу, занадто великої кількості людей на майданчику або через неможливість купування квитка в касах закладу завчасно. В такій ситуації заощадити час та не забувати про власне здоров'я може допомогти веб-додаток.

Отже, проблема полягає в тому аби надати можливість студенту завчасно забронювати спортивні заняття за наявності вільних місць на майданчику.

Тому актуальним є створення ПЗ, в якому користувач матиме можливість віддалено забронювати чи відмінити бронювання місця на секції, купити квиток та отримати його електронну версію. Користувачами системи можуть бути студенти ЗВО, де встановлений майданчик, а також інші жителі міста.

Система відкритого спортивного майданчику складається з мобільного додатку та веб-додатку, серверу, який оброблює основні дані системи, та серверу STOP-NET, який надсилає пропускну турнікету дані для проходження певного користувача.

Основа веб-додатку полягає в роботі з сервером, можливості зареєструватись чи авторизуватись в системі, отримати дані щодо розкладу, занять, кількості вільних місць та деталей події, записатись і отримати квиток на заняття.

На сьогодні існують схожі рішення, які дозволяють людям бронювати заняття на різноманітні види спорту до певних спортивних установ, наприклад, «Спортлайф», «Атлетіко». Але дана система дозволить студентів записуватись на заняття на майданчику, який знаходиться на території університету. Це означає, що користувач зможе відвідувати найближчий майданчик, отримувати квиток, який для студентів є безкоштовним, та не витрачати свій час, в випадку якщо по його приходу там не було вільних місць. Також це допоможе студентам реалізувати план спортивно-оздоровчих заходів університету.

Таким чином, розроблювана система дозволить користувачеві відслідковувати заходи, які проходять на майданчику, та бути його активним учасником.

ДЖЕРЕЛА

1. Гуменний В. Особливості фізичного виховання студентів вищих навчальних закладів на основі урахування специфіки професійної діяльності / В. Гуменний В // Спортивний вісник Придніпров'я. – 2013.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVII Міжнародної
науково-практичної конференції
молодих вчених та студентів
м. Київ, 23-26 квітня 2019 року,

ТОМ 2



Київ- 2019

Використання методів виявлення автоматичних моделей поведінки для побудови аналітичних сценаріїв.	122
<i>ОЛСНЄВА К.М., аспірант;</i>	
<i>ШПУРИК В.В.</i>	
<i>Керівник - доц., к.т.н. Коваль О.В.</i>	
Візуалізації структури KNX-мережі з використанням людинно-машинного інтерфейсу.	123
<i>ЯШИН А.С., магістрант гр. ТМ-81мп</i>	
<i>Керівник - доц., к.е.н. Сегеда І.В.</i>	
Розробка агента моніторингу і управління попитом на електричну енергію "Розумної будівлі".	124
<i>ШАРПІН С.А., магістрант гр. ТІ-81мп</i>	
<i>Керівник - доц., к.т.н. Ковальчук А.М.</i>	
Агент моніторингу та управління режимами роботи мікроенергостанцій.	125
<i>СТОЛЯР А.В., магістрант гр. ТВ-81мп</i>	
<i>Керівник - доц., к.т.н. Ковальчук А.М.</i>	
Розробка серверної частини для веб-додатку відкритий спортивний майданчик з е-сервісами.	126
<i>СЕРБІН А.В., магістрант гр. ТВ-82</i>	
<i>Керівник - доц., к.т.н. Ковальчук А.М.</i>	
Розв'язок задачі обрахунку водонагрівача в інтерактивному режимі з використанням клієнт-серверної архітектури.	127
<i>РОМАНОВ О.В., магістрант гр. ТВ-61мп</i>	
<i>Керівник - доц., к.т.н. Кузьменко І.М.</i>	
Оцінка територій для побудови вітроелектростанцій із застосуванням мультіагентних технологій та ПС.	128
<i>ПІДВИШЕННИЙ Т.О., магістрант гр. ТВ-81мп</i>	
<i>Керівник - асист. Швайко В.Г.</i>	
REST-інтерфейс як основа комунікацій систем контролю доступу.	129
<i>ОБРУСНИК Д.В., студент гр. ТВ-82мп</i>	
<i>Керівник - доц., к.е.н. Левченко Л.О.</i>	
Інструментальні засоби аналізу впливу параметрів експерименту на сигнатуру морського об'єкту.	130
<i>ОБЩЕНКО А.А., магістрант гр. ТВ-81мп</i>	
<i>Керівник - доц., к.т.н. Варава І.А.</i>	
Система моделювання структури та функціонального контенту інженерних систем енергоефективної будівлі.	131
<i>КУРСЕНКО Л.О., магістрант гр. ТІ-81мп</i>	
<i>Керівник - доц., к.т.н. Шпурик В.В.</i>	
Інструментальний засіб підтримки динамічного реєстру інформаційних ресурсів на базі ОРБД Caché.	132
<i>КОСТЕНКО О.П., магістрант гр. ТВ-81мп</i>	
<i>Керівник - доц., к.т.н. Кублій Л.І.</i>	
Мультіагентні системи в децентралізованих мережах енергоспоживання.	133
<i>ЖОРНОВИЙ Е.Г., магістрант гр. ТВ-82</i>	
<i>Керівник - доц., к.т.н. Ковальчук А.М.</i>	
Розробка веб-додатку для відкритого спортивного майданчику з е-сервісами.	134
<i>АМБРОС С.М., магістрант гр. ТВ-81мп</i>	
<i>Керівник - доц., к.т.н. Ковальчук А.М.</i>	

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ВІДКРИТОГО СПОРТИВНОГО МАЙДАНЧИКУ З Е-СЕРВІСАМИ

Невелика кількість спортивних майданчиків, їх стан впливають на недостатню активність молоді в спортивних заходах. Сучасно облаштований майданчик з можливістю електронного доступу до актуальної інформації щодо його роботи, допоміг би дізнаватись про заходи майданчику в онлайн режимі.

Студенти полюбляють спорт, але часом через навантажений навчанням або певними проблемами графік, вони можуть не встигати відвідувати спортивні секції, через незнання розкладу, занадто великої кількості людей на майданчику або через неможливість купування квитка в касах закладу завчасно. В такій ситуації заощадити час та не забувати про власне здоров'я може допомогти веб-додаток.

Отже, проблема полягає в тому аби надати можливість студенту завчасно забронювати спортивні заняття в підходящий час за наявності вільних місць на майданчику.

Тому актуальним є створення ПЗ в якому користувач матиме можливість віддалено забронювати чи відмінити бронювання місця на секції, купити квиток та отримати його електронну версію. Користувачами системи можуть бути студенти вузу, де встановлений майданчик, а також інші жителі міста.

Система відкритого спортивного майданчику складатиметься з мобільного додатку та веб-додатку, серверу, який оброблює основні дані системи, та серверу STOP-NET, який надсилає пропускну турнікету дані для проходження певного користувача.

Основа веб-додатку полягає в роботі з сервером, можливості зареєструватись чи авторизуватись в системі, отримати дані щодо розкладу, занять, кількості вільних місць та деталей події, записатись чи відмінити заняття, отримати квиток на заняття.

На сьогодні існують схожі рішення, які дозволяють людям бронювати заняття на різноманітні види спорту до певних спортивних установ, наприклад, спортлайф, атлетіко. Але дана система дозволить студентів записуватись на заняття на майданчику, який знаходиться на території університету. Це означає, що користувач зможе відвідувати найближчий майданчик, отримувати квиток, який для студентів є безкоштовним, та не витрачати свій час, в випадку якби по його приходу там не було вільних місць. Також це допоможе студентам реалізувати план спортивно-оздоровчих заходів університету.

Вхідними даними для веб-додатку є особисті дані користувача, а саме ім'я, прізвище, пошта, студентський квиток(якщо він є), вид спорту або бажаний час заняття. Вихідними даними є розклад з наявністю вільних місць та згенерований унікальний квиток (в випадку бронювання заняття), який може бути збережений та переглянутий в особистому кабінеті користувача.

Таким чином, розроблювана система дозволить користувачеві відслідковувати заходи, які проходитимуть на майданчику та бути його активним учасником.

Перелік посилань:

1. Гуменний В. Особливості фізичного виховання студентів вищих навчальних закладів на основі урахування специфіки професійної діяльності / Гуменний В. // Спортивний вісник Придніпров'я. – 2013.